FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Mordomo - Sistema de Gestão de Habitação

Emanuel Barbosa da Silva Tiago

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Mário Jorge Rodrigues de Sousa

Coorientador: Eng.º José Morgado

julho de 2016

Resumo

O mercado da domótica tem tido um acentuado crescimento nos últimos anos derivado do aparecimento do conceito *Internet of Things*, que visa dotar dispositivos físicos, carros, casas e outros objetos com sensores, *software*, eletrónica e ligação à internet, a fim de comunicarem e trocarem informação uns com os outros autonomamente, formando um sistema com inteligência própria. Esta evolução é consequência da atenção recente que grandes empresas mundiais como por exemplo a Samsung, a Microsoft ou a LG têm dado a esta área colocando novas soluções no mercado. Desta forma, as soluções mais completas apresentam conjuntos de *software* e *hardware* interligados que fornecem um enorme leque de funcionalidades para controlar, monitorizar e gerir o espaço residencial. A evolução da domótica aparece também como resultado do crescimento do mercado dos *smartphones*, que cada vez mais apresentam especificações técnicas tecnologicamente evoluídas.

Como tal, o objetivo desta dissertação passa precisamente pela especificação, desenho e desenvolvimento de um sistema piloto na área da domótica, investigando todos os componentes do sistema desde a aplicação móvel executada no *smartphone* que é o principal meio de interação com o utilizador, até ao módulo de acesso ao *hardware* que controla um ou mais dispositivos, representado neste projeto com recurso a um Raspberry Pi. A criação deste sistema tem como principal objetivo o estudo e avaliação das capacidades disponibilizadas pelos diferentes componentes do sistema, analisando a implementação de determinadas funções nos *smartphones* e no Raspberry Pi, para futuramente ser projetado um sistema capaz de ser comercializado.

Assim sendo, a implementação passa pelo desenvolvimento de cinco aplicações: uma aplicação móvel, um *web service*, uma aplicação de controlo autónomo, um módulo de acesso a base de dados e outro de acesso à interface GPIO do Raspberry Pi. A aplicação móvel é o principal meio de interação com os utilizadores, apresentando os dados dos mesmos e disponibilizando controlos para execução de tarefas. O *web service* garante as comunicações realizadas entre as diversas aplicações do sistema, nomeadamente com a base de dados e com a interface GPIO. A aplicação de controlo autónomo consulta as regras implementadas e tem como objetivo garantir a atuação nos dispositivos segundo as mesmas. O módulo de acesso à base de dados funciona para inserir e ler informação partindo da base de dados associada ao sistema, e o módulo de acesso à interface GPIO garante a atuação nos dispositivos adicionados e ligados aos pinos.

O projeto incorporou um plano de testes pensado inicialmente, ao qual se submeteu a implementação. Os resultados deste plano de teste foram positivos mostrando o sucesso da implementação efetuada e o cumprimento dos objetivos.

As principais ferramentas utilizadas foram o IDE Visual Studio, com a plataforma Xamarin e a API Xamarin.Forms, para garantir a operabilidade nos três principais sistemas operativos móveis. A base de dados foi implementada segundo o sistema PostgreSQL.

Abstract

The home automation market has grown significantly over the recent years because of the emergence of Internet of Things concept, which aim to grant physical devices, vehicles, buildings and other objects with sensors, software, eletronics and network connection, so they can communicate and exchange data with each other by themselves, creating a system with his own intelligence. This evolution is a consequence of the attention that big companies like Samsung, Microsoft or LG are giving to this market, developing new solutions. In this way, the most complete solutions present software and hardware combined into a system, offering a huge number of funcionalities to control, monitor and manage houses. The evolution of home automation results from the growth of the mobile phone market as well, which are more powerful every day.

As such, the goal of this dissertation is to specify, design and develop an home automation pilot system, studying all the components of the system from the mobile application running on a smartphone which is the main interaction point with the user, till the module of access to hardware which controls one or more devices, represented in this project with the usage of a Raspberry Pi. The criation of the system has his main goal on the study and evaluation on the capacities offered by the different parts, analyzing the implementation of specific functions on smartphones and Raspberry Pi, so in the future a system can be conceived to be comercialized.

Thus, the implementation goes through the development of five different applications: one mobile phone application, a web service, an application to apply autonomous control, a module to access the database and a module to access the GPIO pinout. The mobile phone application is the main interaction point with the user, displaying their data and giving controls to execute some tasks. The web service ensure that communications between the different applications can be done, mainly with the database and the GPIO interface. The autonomous control application checks all the specified rules and guarantees the control on the connected devices. The module to access the databse serves to insert and get information from the database, as the module to access the GPIO pinout interacts directly with the devices.

The project had a test plan, used to test the implementation. Tbe results were positive showing the success of the implementation as well as the fulfillment of the specified objectives.

The main tools used were the Visual Studio IDE, with the Xamarin plataform and Xamarin.Forms API, which ensure the operability on the three main mobile operating systems. The database was implemented using the PostgreSQL service.

Agradecimentos

Este espaço serve para agradecer a todos que uma ou de outra forma me apoiaram para o sucesso deste projeto, que para além de requerer um esforço intelectual também exige um esforço a nível pessoal, e como tal afeta outras pessoas para além de mim que diariamente interagem comigo.

Primeiramente, queria agradecer à CimSoft - Tecnologias de Informação, Lda e ao meu orientador Eng. José Morgado, pelo acolhimento nas suas instalações, facultando todas as condições para que o projeto fosse exequível.

Agradeço ao meu orientador, Professor Mário de Sousa, pela sua disponibilidade e ajuda durante a realização desta dissertação.

Agradeço aos meus amigos e companheiros nesta jornada académica, pelo apoio, humildade e entreajuda que caracterizaram estes cinco ou treze anos de percurso académico conjunto.

Agradeço à Inês, por todo o apoio, paciência e motivação com os quais me presenteou não só neste projeto mas em todas as fases do meu percurso académico.

Agradeço à minha família, principalmente à minha mãe e ao meu pai, pelo seu trabalho e esforço, para me possibilitarem uma formação a este nível. Agradeço o seu apoio e motivação particularmente durante a realização desta dissertação.

Emanuel Barbosa da Silva Tiago

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

IFA	Internationale Funkausstellung Berlim
IDE	Integrated Development Environment
API	Application Programming Interface
SDK	Software Development Kit
BPS	Bits por segundo
EIBA	European Installation Bus Association
EHSA	European Home Systems Association
BCI	BatiBUS Club International
GPIO	General Purpose Input/Output

Capítulo 1

Introdução

Este documento foi elaborado no âmbito da unidade curricular Dissertação, pertencente ao Mestrado Integrado em Engenharia Eletrotécnica e de Computadores.

Neste primeiro capítulo será apresentado o contexto em que surgiu este projeto, os seus objetivos principais, e qual a motivação para se realizar um projeto deste cariz.

1.1 Contexto

O trabalho proposto para dissertação foi desenvolvido em ambiente empresarial, na CimSoft – Tecnologias da Informação, Lda, e enquadra-se dentro de uma área específica do vasto mundo de *software*, mais especificamente na área de aplicações para dispositivos móveis. Considerando o leque de aplicações com funcionalidades e fins distintos, o trabalho a ser desenvolvido está incluído na categoria de aplicações para automação residencial, ou domótica.

A domótica é o conceito utilizado para definir o resultado da junção entre os edifícios e a robótica, tornando-os assim edifícios inteligentes, construções que integram não só o desenho arquitetónico e a sua edificação, como também tecnologias avançadas de comunicação, gestão, segurança, conforto, entre outras. Assim sendo, os edifícios são dotados de processos automáticos e inteligência própria.

A empresa na qual o projeto é desenvolvido apresenta essencialmente dois produtos de *software*: um de gestão documental, principalmente vocacionado para agências de advogados, e um POS (Point-of-Sale), pensado para diversos negócios como mercearias, cafés, lavandarias, hotéis, entre outros. Com base nestes projetos já criados e disponíveis no mercado, a empresa pretende inovar procurando crescer de duas formas aproveitando a era dos *smartphones*:

1. Em primeiro lugar, planeia melhorar o seu *software* POS acrescentando um módulo para dispositivos móveis onde é possível monitorizar alguns aspetos do negócio em questão. Um exemplo disso é o registo de anomalias para posterior análise nos hotéis.

2. Em segundo lugar, procura entrar no mundo da domótica, desenvolvendo um primeiro sistema básico de teste e prova de conceito para avaliar as capacidades do *hardware* (*smartphones* e Raspberry Pi) e *software* (Xamarin), a fim de posteriormente selar uma eventual parceria com empresas já possuidoras dos seus próprios sistemas de domótica, ou com empresas que comercializam produtos para o espaço residencial e assim inovar esses produtos, como por exemplo equipamento de cozinha, sala, e assim por diante.

É com base nestes objetivos da empresa e com recurso às novas capacidades dos *smartphones*, que permitem desenvolver aplicações para melhor controlar e gerir o espaço que é o nosso lar, que aparece este projeto.

1.2 Objetivos

O principal objetivo desta dissertação é o desenvolvimento de uma aplicação para dispositivos móveis, possível de ser executada nos principais sistemas operativos móveis, ou seja, Android e iOS, e se possível no Windows 10 Mobile, a partir de qualquer local. Esta terá em vista a automação e controlo de uma casa e monitorização das atividades. O *software* desenvolvido irá comunicar com componentes de *hardware* existentes no mercado. A par da aplicação executável nos dispositivos móveis, é fundamental existir um meio de acesso *back office* para administração do sistema: manutenção, gestão e suporte à aplicação móvel.

O sistema deverá ter ligação a uma base de dados. Nesta base de dados são guardadas todas as informações indispensáveis ao funcionamento da aplicação, nomeadamente os utilizadores, associados a determinados dispositivos, regras, eventos, entre outros. Aproveitando a base de dados, à aplicação móvel aplicar-se-á um sistema de autenticação.

Funcionalmente, a aplicação irá incluir quatro grandes capacidades:

- Registo de eventos para posterior observação e monitorização;
- Controlo do ambiente com base em regras definidas previamente: ler a situação atual, consultar as regras existentes e por fim atuar sobre os dispositivos;
- Controlo do ambiente a partir de decisões tomadas no momento e enviadas através da aplicação móvel;
- Programação de atividades para um determinado dispositivo.

A comunicação a decorrer entre a aplicação do telemóvel e os dispositivos a controlar é feita através de um Raspberry Pi, que representa a camada de acesso ao *hardware*, e que corre simultâneamente um *web service*, o qual se encarrega de garantir a partilha de informação entre todos os componentes do sistema, nomeadamente leitura e escrita na base de dados.

O sistema no que diz respeito a requisitos não funcionais deverá garantir um funcionamento fluído e *user friendly*, ou seja, visualmente é exigido que a aplicação apresente uma interface fácil de utilizar, apelativa e intuitiva de manipular; para além disto, a integridade de todas as

informações e comunicações deve ser assegurada na eventualidade de falhas de energia ou de avaria de um componente do sistema.

1.3 Motivação

Graças ao acentuado desenvolvimento tecnológico que assistimos na área dos dispositivos móveis, nomeadamente smartphones e tablets, nos dias que correm as respetivas aplicações atingem níveis de sofisticação tão elevados que permitem às pessoas manter no seu quotidiano comunicação constante entre si e com aparelhos eletrónicos, garantindo o controlo dos mesmos, a curtas ou longas distâncias. Consequentemente, a automação residencial, essencialmente para gestão da energia e conforto, começa a emergir no mercado, e as soluções apresentam funcionalidades cada vez mais evoluídas, tecnologicamente complexas, e úteis do ponto de vista do utilizador.

Empresas relevantes no mercado tecnológico, como é exemplo a Samsung, começam a apostar cada vez mais em soluções de domótica, como foi possível acompanhar no evento IFA[16].

A motivação prende-se, precisamente, com trabalhar numa vertente onde está prevista uma grande margem de crescimento, altamente procurada pela sociedade atual, apresentando um produto inovador, capaz de oferecer novos meios de controlo da sua casa às pessoas interessadas.

1.4 Estrutura do documento

Para além deste capítulo introdutório, este documento apresenta outros cinco capítulos. No capítulo 2 são apresentadas tecnologias atuais no mundo da domótica, a par de soluções comerciais e *open-source*. No capítulo 3 é explicado o problema, e a abordagem considerada para solucionar o problema. Seguidamente, no capítulo 4 são explicados todos os procedimentos relacionados com a implementação da solução, sendo este seguido pelo capítulo 5 onde são demonstrados os resultados obtidos. Por último, no capítulo 6 é feita uma conclusão, com análise do trabalho realizado e sugestões para trabalho futuro.

Capítulo 2

Revisão Bibliográfica

Este capítulo tem como objetivo demonstrar as diferentes tecnologias estudadas para compreensão e integração na área da Domótica. Irão ser descritos os protocolos mais utilizados nos mercados Europeu e Norte-Americano respetivos ao hardware como, por exemplo, os protocolos KNX e X10, e soluções comerciais de maior relevância. Por fim, serão abordados alguns ambientes de desenvolvimento, entre os quais o Xamarin que irá ser utilizado neste projeto, e apresentadas aplicações móveis disponíveis no mercado com fim semelhante ao desejado.

2.1 Protocolos de comunicação

Atualmente, existe um elevado número de protocolos de comunicação criados e desenvolvidos por empresas que pretendiam competir no mercado da domótica. Desta forma, ao utilizador é disponibilizado um vasto leque de soluções para escolha baseados em tecnologias standard e tecnologias proprietárias.

As tecnologias standard nascem geralmente da aliança entre dois ou mais fabricantes; são tecnologias cujo princípio de funcionamento e utilização estão abertos a qualquer pessoa. Estas tecnologias permitem que todas as entidades que desejam possam desenvolver os seus protocolos de comunicação e criar os seus próprios produtos, aumentando assim a oferta e as funcionalidades disponíveis.

Por outro lado, as tecnologias proprietárias têm como base um protocolo fechado, cuja utilização está restrita à empresa desenvolvedora e requer a aquisição dos produtos desenvolvidos por essa mesma empresa.

2.1.1 KNX

O KNX é um protocolo de comunicação constituído por uma pilha protocolar estruturada num conjunto de camadas semelhante ao modelo de camadas OSI, resultante de um processo de convergência entre três standards europeus de automação predial e residencial: EIBA, BCI e EHSA. Este

protocolo é um standard mundial (ISO/IEC 14543-3)[1], aberto a todos que desejam desenvolver na área da domótica. Desta forma, os produtores tem acesso a um meio de comunicação aceite internacionalmente e podem construir aparelhos para as mais diversas funcionalidades, como o controlo luminoso, aquecimento, ventilação, ar condicionado, gestão energética, monitorização, sistemas de alarme, entre outros.

Figura 2.1: Ilustração das diferentes aplicações possíveis com o protocolo KNX[1]

O KNX possui uma arquitetura descentralizada, definindo uma relação elemento a elemento, o que permite distribuir a inteligência entre os sensores e atuadores instalados [17]. Inicialmente este protocolo funcionava sobre um cabo do tipo telefónico; contudo, foi melhorado de forma a funcionar sobre os diferentes meios:

• TP(twisted pair) TP1/TP0: Este meio de comunicação é proveniente da EIBA. Funciona sobre um par de condutores com um bitrate de 9600bps. Ambos os fios fornecem uma alimentação de 24Vdc para os participantes. Utilizando o método de comunicação CSMA-CA para evitar as colisões, maximiza a largura de banda disponível. Este foi o primeiro a ser disponibilizado e é o único a ter relevância.

• PL(powerline) PL110/PL132: Este meio de comunicação é também proveniente da EIBA. Utilizando a rede elétrica de uma casa, com uma transmissão que apresenta um bitrate de 1200bps, este método usa uma modulação SFSK, com um princípio de funcionamento semelhante à modulação FSK mas com maior separação entre as portadoras. A distância permitida sem repetidor é aproximadamente 600 metros.

• RF(rádio frequência): Os dispositivos que suportam este meio de comunicação utilizam sinais de rádio para enviar telegramas KNX. Os telegramas são transmitidos na frequência de banda dos 868MHz para dispositivos com alcance pequeno, com uma potência radiada máxima de 25mW, e um bitrate de 16.384kbps. É caracterizado pelo seu baixo consumo, conseguindo-se cobrir distâncias até 300 metros, sendo possível a utilização de repetidores para maiores distâncias.

• IP (Ethernet): Utiliza a rede Ethernet a 10Mbps, transmitindo telegramas KNX encapsulados em telegramas IP. Desta forma, redes LAN e Internet podem ser usados para envio dos telegramas KNX, permitindo a transferência dos mesmos a grandes distâncias, entre instalações muito afastadas.

• IR (infravermelho): O meio de comunicação é o sinal infravermelho, que permite uma distância máxima de aproximadamente 12 metros. É o ideal para o uso dentro de espaços restritos, onde contém um número de dispositivos KNX instalados limitado e as distâncias são reduzidas.

Com os meios de comunicação descritos previamente disponíveis, o protocolo contempla três modos de funcionamento[2]:

• S-mode (System mode): A configuração do modo sistema usa a mesma filosofia que o EIBA atual, isto é, os diversos dispositivos são instalados e configurados recorrendo a um *software* concebido propositadamente para este fim, o ETS. Este modo de funcionamento oferece um elevado grau de flexibilidade para o desenvolvimento de novas funcionalidades de controlo, sendo o mais utilizado. Os dispositivos S-mode só poderão ser comprados através de distribuidores especializados.

• E-mode (easy mode): A configuração do sistema neste modo requer pessoal com pequeno nível de treino. Os dispositivos são pré-programados na fábrica e carregados com uma definição dos parâmetros; com um simples configurador, cada dispositivo pode ser reconfigurado, principalmente a nível dos parâmetros definidos e das suas ligações de comunicação. Este modo de funcionamento oferece um número de funcionalidades limitado comparando com o S-mode.

• A-mode(automatic mode): A configuração do sistema neste modo automático segue uma filosofia de Plug&Play, na qual nem o instalador nem o utilizador final necessitam de configurar o dispositivo. É especialmente indicado para uso em eletrodomésticos e equipamentos de entretenimento (consolas, set-top boxes, áudio e vídeo).

O seguinte esquema apresenta o funcionamento descrito anteriormente:

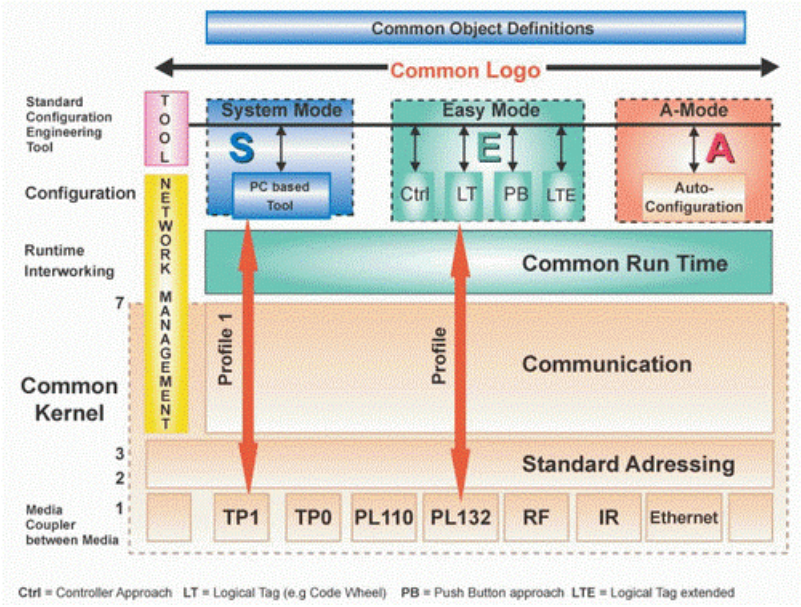

Figura 2.2: Esquema representativo das diferentes camadas do protocolo KNX[2]

2.1.2 X10

O X10 é um protocolo de comunicação entre dispositivos eletrónicos utilizado na área da domótica. Este é bastante usado a nível mundial por não requerer nenhuma instalação adicional, ou seja, tira partido da rede elétrica instalada na casa que, em Portugal funciona a 230Vac, para sinalização e controlo e pode ser aplicado a qualquer momento, aquando da construção bem como posteriormente.

Os sinais enviados segundo este protocolo envolvem pequenas rajadas de sinais de rádio frequência que contêm informação digital [3].

No protocolo X10 as transmissões são sincronizadas com a rede elétrica instalada; isto indica que estas ocorrem aquando da passagem pelo valor zero por parte do valor da rede AC (denominado crossing point). O objetivo é que, idealmente, as transmissões ocorram precisamente no instante em que o valor é zero, sendo o desvio máximo permitido igual a 200 microssegundos.

Relativamente ao envio de informação, esta é enviada em sinal binário, o que se traduz numa representação com “0” e “1”. O valor “1” é demonstrado por uma rajada de 1 milissegundo a 120kHz no crossing point, sendo o valor “0” demonstrado pela ausência desta rajada. Assim sendo, estes valores devem ser enviados três vezes cada um para coincidir com o crossing point das três fases do sinal elétrico.

O envio completo do código exige a transmissão ao longo de onze ciclos. Os dois primeiros ciclos representam o Start Code; os quatro próximos ciclos representam o House Code; e, por fim, os últimos cinco ciclos representam o Number Code ou o Function Code. Estes códigos que juntos

formam um código completo devem ser enviados sempre em grupos de dois sendo que, entre cada grupo, é necessário existir três ciclos sem envio de informação.

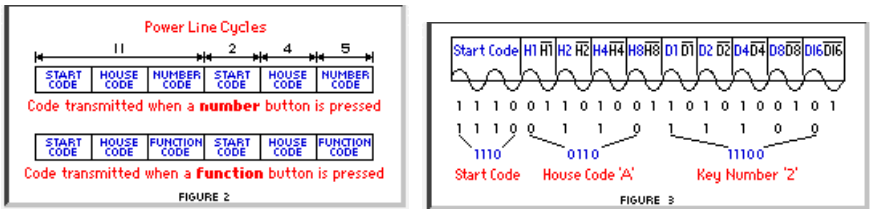

Figura 2.3: Representação gráfica dos códigos enviados no protocolo X10[3]

No envio de um bloco de informação, ou seja, conjuntos de quatro bits (House Code) ou cinco bits (Number Code ou Function Code), devem ser transmitidos os seus complementos. Este mecanismo funciona da seguinte forma: no momento do envio do valor "1" na primeira metade do ciclo, na segunda metade do ciclo não é enviado sinal, representando o valor "0", e vice-versa.

A seguinte tabela mostra os conjuntos de bits que podem ser enviados para cada parcela do código, sendo que, o Start Code é sempre composto pela sequência 1110.

HOUSE CODES	H1	H2	H4	H8
A	0	1	1	0
B	1	1	1	0
C	0	0	1	0
D	1	0	1	0
E	0	0	0	1
F	1	0	0	1
G	0	1	0	1
H	1	1	0	1
I	0	1	1	1
J	1	1	1	1
K	0	0	1	1
L	1	0	1	1
M	0	0	0	0
N	1	0	0	0
O	0	1	0	0
P	1	1	0	0

KEY CODES	D1	D2	D4	D8	D16
1	0	1	1	0	0
2	1	1	1	0	0
3	0	0	1	0	0
4	1	0	1	0	0
5	0	0	0	1	0
6	1	0	0	1	0
7	0	1	0	1	0
8	1	1	0	1	0
9	0	1	1	1	0
10	1	1	1	1	0
11	0	0	1	1	0
12	1	0	1	1	0
13	0	0	0	0	0
14	1	0	0	0	0
15	0	1	0	0	0
16	1	1	0	0	0
All Units Off	0	0	0	0	1
All Lights On	0	0	0	1	1
On	0	0	1	0	1
Off	0	0	1	1	1
Dim	0	1	0	0	1
Bright	0	1	0	1	1
All Lights Off	0	1	1	0	1
Extended Code	0	1	1	1	1
Hail Request	1	0	0	0	1①
Hail Acknowledge	1	0	0	1	1
Pre-Set Dim	1	0	1	X	1②
Extended Data (analog)	1	1	0	0	1③
Status=on	1	1	0	1	1
Status=off	1	1	1	0	1
Status Request	1	1	1	1	1

FIGURE 4

Figura 2.4: Possíveis códigos enviados em cada parâmetro no protocolo X10[3]

O principal problema em comunicar utilizando a rede elétrica é a suscetibilidade aos ruídos que possam existir na mesma. Os ruídos são sinais elétricos indesejados que se encontram na rede a par dos sinais desejados. Assim sendo, para reduzir a possibilidade de os sinais serem afetados por ruídos, no protocolo X10 instaurou-se os dois procedimentos previamente referidos: a modelação das ondas para os 120kHz e envio do complemento nos códigos de informação[8].

Este protocolo representa uma taxa de transmissão reduzida, aproximadamente 50bps (é enviado um bit a cada ciclo da rede elétrica, que na Europa funciona a 50Hz). Uma consequência direta deste fator é a impossibilidade de utilizar este protocolo para transmissões de sinais digitais de alta resolução como sinais de vídeo, televisão e hi-fi, sendo aconselhado apenas para envio de

sinais simples compostos por poucos dados, como ligar ou desligar aparelhos, luzes e regular a intensidade luminosa de uma lâmpada.

A título de curiosidade, foi concebida para o ambiente de desenvolvimento do microprocessador Arduíno uma biblioteca para possibilitar a comunicação entre o Arduíno e os dispositivos presentes no edifício segundo o protocolo X10 [18], aumentado o número de possíveis utilizações e a facilidade de implementação de funcionalidades utilizando este protocolo.

2.1.3 INSTEON

O INSTEON é um protocolo que permite estabelecer comunicações entre dispositivos pouco avançados tecnologicamente e de baixo custo.

Este protocolo funciona sobre uma rede denominada *dual-mesh*, cuja designação indica que este utiliza as ondas de rádio frequência bem como a ligação elétrica instalada no edifício para garantir as comunicações entre os dispositivos. É fundamental referir que, nas comunicações realizadas através da instalação elétrica da casa, o protocolo INSTEON tira partido dos princípios da rede X10 descrita previamente.

O funcionamento deste protocolo distingui-se no seguinte aspeto: todos os dispositivos presentes na rede funcionam como recetores e como transmissores e estão ligados *peer-to-peer*, ou seja, qualquer dispositivo pode transmitir, receber ou repetir mensagens sem ter que pedir permissão. Este princípio ajuda a que o protocolo INSTEON apresente uma consequência oposta ao comum a partir do momento em que há um aumento do número de dispositivos na rede, isto é, em vez de a rede se tornar mais sobrecarregada e menos fiável, esta torna-se mais robusta, pois quanto mais dispositivos forem colocados na rede mais certo será que as mensagens cheguem ao seu destino e com melhor intensidade de sinal. De uma forma geral, os protocolos utilizam apenas um ponto central, nomeadamente, um router que está encarregue de repetir as mensagens. Este princípio de funcionamento permite também que um dispositivo capaz de comunicar apenas por ondas de rádio frequência receba ou envie comandos para um dispositivo capaz de comunicar unicamente por corrente elétrica; para isto, é necessário apenas que entre ambos exista um dispositivo capaz de comunicar através das duas tecnologias.

Obviamente, este mecanismo de repetição de mensagens pode originar tempestades de informação de repetições infinitas da mesma mensagem. Para contrariar esta adversidade, o protocolo estabelece três restrições fundamentais: em primeiro lugar, todos os dispositivos emitem a informação dentro de um intervalo de tempo, por outro lado, para cada mensagem recebida no seu envio são definidos dois valores: *Max Hops*, que é o número máximo de vezes que a mensagem pode ser retransmitida, sendo este valor nunca superior a 3, e o *Hops Left* que é o número de retransmissões que restam para essa mensagem.

Todas as mensagens são confirmadas no momento da sua receção, e na ocorrência de qualquer erro as mensagens são reenviadas automaticamente[4]. Estas confirmações são dadas pelo destino da mensagem, ou seja, se a mensagem for recebida pelo dispositivo a quem a mensagem era destinada, este envia uma mensagem específica denominada *acknowledgement* depois de repetir

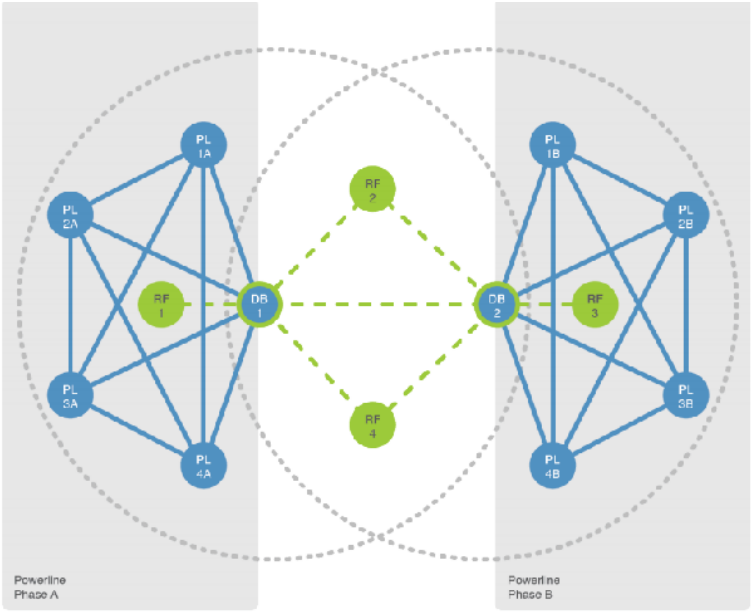

Figura 2.5: Funcionamento do protocolo INSTEON[4]

a mensagem recebida pela última vez. Esta mensagem chamada *acknowledgement* é repetida o mesmo número de vezes que a mensagem original.

2.1.4 Zigbee

O Zigbee é um protocolo de comunicação entre dispositivos eletrónicos que se destaca essencialmente em três aspetos: baixo consumo de potência, baixa taxa de transmissão de dados e baixo custo de implementação. Este protocolo respeita a definição de camadas segundo o modelo OSI.

A tecnologia utilizada é semelhante às redes Wi-Fi e Bluetooth, diferenciando-se destas pelo seu menor consumo energético, bem como pelo alcance permitido mais reduzido (cerca de 10 metros)[19]. Em contrapartida, esta tecnologia permite que todos os intervientes na rede retransmitam a informação, fazendo com que a comunicação entre duas unidades seja garantida pela repetição sucessiva da mensagem pelos intervientes na rede até a entrega ser bem-sucedida.

Neste protocolo as mensagens são enviadas recorrendo a sinais de radiofrequência. O sistema está especificado para funcionar sobre uma das seguintes três larguras de banda disponíveis: 2.4GHz e 915MHz na América do Norte e 868MHz na Europa. Para estas larguras de banda, o sistema é capaz de atingir frequências de envio de informação iguais a 250kbps, 40kbps e 20kbps, respetivamente. Com o intuito de evitar colisões e manter o bom funcionamento em situações onde o ambiente de comunicação está congestionado, o protocolo implementa técnicas CSMA.

Funcionalmente, um dos pontos mais importantes é o fato de a tecnologia Zigbee utilizar uma estruturação *mesh network*, ou melhor, uma rede desenhada em malha. A primeira grande vantagem deste método é o aumento de fiabilidade da rede, pois assumindo que a mensagem tem múltiplos caminhos que pode utilizar simultâneamente para atingir o seu destino, a probabilidade de a mensagem ser corretamente recebida é superior; a segunda vantagem é a extensão de rede

capaz de ser obtida, pois como as mensagens são retransmitidas entre dispositivos é possível que uma mensagem seja enviada de um emissor para um recetor com intervientes entre os mesmos, não necessitando de uma interação direta entre ambos.

2.1.5 Z-Wave

O protocolo Z-Wave tem um príncipio de funcionamento semelhante ao Zigbee. Befinicia da mesma estrutura de implementação e comunicação entre dispositivos, assim como apresenta as mesmas adversidades. Contudo, ao contrário da Zigbee, a tecnologia Z-Wave utiliza as bandas de frequência 915MHz nos Estados Unidos, e 868MHz na Europa. Consequentemente, os alcances máximos são superiores na utilização do protocolo Z-Wave, iguais a 30 metros. Para além disto, a tecnologia Z-Wave utiliza modelação FSK, enquanto a tecnologia Zigbee utiliza modelação OQPSK.

Como resultado das diferenças entre os métodos operacionais de ambas as tecnologias, a Zigbee tem aplicabilidade principalmente na domótica, *smart grids* e controlo remoto, enquanto que a Z-Wave tem na domótica e segurança.

2.1.6 Comparação de protocolos

A seguinte tabela apresenta uma comparação dos protolocos comercialmente mais requisitados do grupo de protocolos descritos previamente.

	X10	ZigBee	Z-Wave	INSTEON
Dual Mesh(Powerline e Wireless RF)	Não	Não	Não	Sim
Powerline Communication	Sim	Não	Não	Sim
Powerline Data Rate(bps)	20	N.A.	N.A.	2.400
RF Wireless Communication	Não	Sim	Sim	Sim
RF Wireless Data Rate(bps)	N.A.	20.000	9.600	38.400
Full Mesh Network Communication	Não	Não	Não	Sim
Automatic Network Enrollment	Não	Não	Não	Sim
Maximum Devices Per Network	256	65.536	232	16,777,216
All Devices are Peers	Não	Não	Não	Sim
Command Acknowledgement	Não	Sim	Sim	Sim
X10 Compatible	Sim	Não	Não	Sim

Tabela 2.1: Comparação entre protocolos[8]

2.2 Sistemas de domótica comerciais

A maioria das soluções comerciais disponíveis no mercado integram as duas componentes do sistema: *hardware* e *software*. Isto implica que quando queremos utilizar a aplicação móvel de um determinado fabricante somos obrigados a comprar os seus meios físicos para estabelecer comunicação com a aplicação.

O número de sistemas de domótica disponíveis no mercado é bastante elevado. Alguns exemplos de sistemas de domótica internacionais são os seguintes:

1. HomeSeer
2. Control4
3. Crestron
4. Vera
5. Staples Connect
6. Iris
7. Svant
8. SmartThings
9. Wink

Os diferentes sistemas de domótica referidos apresentam funcionalidades similares: um controlador central que gere todo o funcionamento da rede de domótica, gestão energética, controlo de sistemas luminosos, termostatos, persianas, sensores de ambiente (luminosidade, humidade, fumo, temperatura), fechaduras, portões de garagem, câmaras de segurança e sensores de movimento. Apenas dois dos sistemas referidos, o Crestron e o Savant, possuem mecanismos de controlo de sistemas audiovisuais. Os protocolos utilizados pelos sistemas são essencialmente o Wi-Fi, Z-Wave e Zigbee; contudo, existem dois que utilizam KNX e Insteon nativamente, como é o caso do Vera e do HomeSeer. Todos os sistemas disponibilizam aplicações para Android e iOS.

Seguidamente vão ser estudados com mais cuidado apenas alguns dos sistemas encontrados.

2.2.1 WeMo

O sistema WeMo disponível no mercado é um conjunto de produtos desenvolvidos pela empresa Belkin International que permite controlar os aparelhos eletrónicos de casa a partir de qualquer local recorrendo a uma aplicação, que funciona até com controlos vocais. O WeMo apresenta 4 principais componentes:

- *WeMo Motion Sensor*: é um sensor que permite desligar e ligar dipositivos eletrónicos assim que as pessoas passem próximo dele; este componente implica que os aparelhos eletrónicos a controlar estejam ligados à mesma rede WiFi;
- *WeMo Insight Switch*: é um interruptor que disponibiliza o consumo de energia e uma estimação de custos relativos ao aparelho conectado a ele;
- *Light Switch*: é um interruptor que permite controlar uma lâmpada;

- *WeMo Switch*: é um interruptor que permite controlar qualquer dispositivo eletrónico que esteja ligado a ele, a partir de qualquer local através de rede WiFi, 3G ou 4G.

Para comunicar com todos os componentes mencionados, é necessário fazer *download* da aplicação móvel chamada *WeMo App*.

Figura 2.6: Principais aplicações do sistema comercial WeMo[5]

O *WeMo Switch* é o dispositivo que oferece mais funcionalidades na integração com a aplicação e meios de controlo dos aparelhos eletrónicos mais interessantes e distintos.

Para além da aplicação óbvia que é o controlo de uma lâmpada, a possibilidade de controlar o dispositivo a partir de qualquer lugar através da utilização da rede 3G ou 4G permite uma interação bastante interessante com alguns aparelhos, nomeadamente com o carregador de telemóvel, com a luz de dormir do filho ou filha, com a máquina de café, entre outros [20].

A Internet das Coisas, ou como é internacionalmente denominada a *Internet of Things (IoT)*, está diretamente relacionada não apenas com o controlo remoto de aparelhos eletrónicos, mas sim com a capacidade de comunicação entre eles que os mesmos possam adquirir. Neste sentido, o sistema WeMo tem um serviço chamado *If This Then That (IFTTT)* que permite definir receitas para garantir a comunicação entre dispositivos. Com estas receitas, é possível executar rotinas, reagir a eventos e passar comandos entre dipositivos; desta forma, pode-se fazer com que os aparelhos se controlem mutuamente. Utilizando o sistema de receitas *IFTTT*, é possível definir ações para os aparelhos eletrónicos baseadas em tempo, como ligar e desligar luzes a uma determinada hora.

Outra funcionalidade inovadora é a possibilidade de atribuir ações com base na localização GPS do dispositivo móvel de controlo; com isto, é realizável desligar parcialmente ou na totalidade as luzes de casa quando se sai.

2.2.1.1 Arquitetura do sistema

O sistema WeMo assegura as suas funcionalidades unicamente através da rede Wi-Fi presenta nas casas dos utilizadores. Assim, a rede de comunicação do sistema WeMo torna-se uma rede

em forma de estrela, onde o router que suporta a rede Wi-Fi é o responsável por assegurar as comunicações entre aparelhos na mesma rede.

O funcionamento descrito anteriormente promove a desnecessidade de incorporar um *hub* como ponto central do sistema, e como os seus aparelhos são compatíveis com a rede Wi-Fi, estes começam a ser explorados por outros sistemas de domótica como é o caso da SmartThings.

2.2.2 SmartThings

A SmartThings é um sistema de controlo residencial recentemente adquirido pela Samsung.

Assim, como muitos outros sistemas, o SmartThings permite controlar uma casa a partir de uma aplicação móvel recorrendo a sensores e a um *hub* que comunicando com os sensores permite interagir com os diferentes aparelhos eletrónicos. Neste sentido, o pacote que é vendido para o sistema SmartThings vem com um *hub* e múltiplos sensores diferentes para poder monitorizar diversas variáveis[21]. O processo de definição de regras e controlo é executado através de uma aplicação móvel.

O *hub* vendido é uma peça fundamental do sistema. Tem uma capacidade de processamento elevada e até consegue funcionar sem energia elétrica da casa; mesmo que o fornecimento elétrico falhe, o *hub* dispõe de um conjunto de baterias AA que disponibilizam energia para aproximadamente 10 horas de funcionamento. O *hub* suporta vídeo, em integração com o sistema de vigilância da Samsung.

Um dos principais e diferenciadores objetivos do sistema é dar aos utilizadores a capacidade de criar os seus próprios métodos ou funções de controlo; o conceito é o seguinte: o utilizador cria a função que deseja que a SmartThings encarrega-se que funcione. Para além de os utilizadores poderem escolher entre diversos aparelhos de diferentes marcas que operam sobre protocolos distintos, os utilizadores têm a possibilidade de desenvolver pequenas aplicações dentro da aplicação inicial fornecida gratuitamente; isto tem como objetivo poder controlar divisões da casa de forma independente, como mostra a seguinte figura:

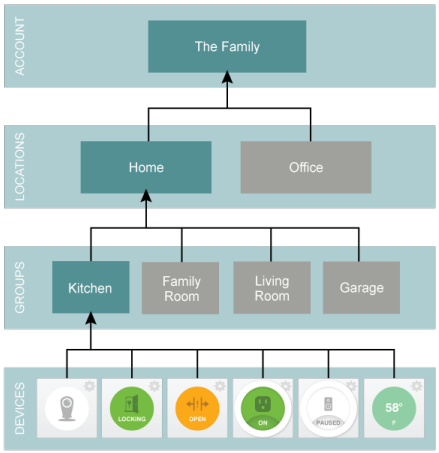

Figura 2.7: Distribuição da aplicação SmartThings por divisões da casa[6]

Como consequência da parceria com a Samsung, existem as vantagens óbvias relativas ao orçamento para efeitos de marketing, para além da eficiência de produção e de distribuição na qual a Samsung é experiente e exemplar. Contudo novas oportunidades surgem, como por exemplo a possibilidade de integração do *hub* do sistema SmartThings em aparelhos distribuídos pela Samsung, como televisões, frigoríficos, máquinas de lavar, entre outros. Esta conjugação de oferta pode resultar numa aliciente oportunidade de compra para os clientes.

Assim, o futuro da SmartThings pode incidir na manutenção e melhoria da liberdade oferecida aos clientes para escolha e desenvolvimento próprio, deixando a produção e conceção de *hardware* para a Samsung.

2.2.2.1 Arquitetura do sistema

Como ponto inicial, é fundamental compreender que neste sistema é sempre tentado separar a inteligência dos dispositivos conectados, deixando assim os dispositivos apenas responsáveis por executar as suas tarefas nativas, como abrir/fechar, ligar/desligar, aquecer/arrefecer, entre outras.

A aplicação funciona com dispositivos pré-definidos, ou seja, conforme o tipo de dispositivo adicionado este vai resultar num conjunto de atributos e comandos normalizados para o mesmo.

A figura 2.8 ilustra a arquitetura utilizada nos sistemas de domótica SmartThings.

Analisando a figura, observamos diferentes camadas que comunicam entre si e garantem o funcionamento do sistema. Estas desempenham as seguintes funções:

- Devices. Estes sãos os meios com os quais o sistema interage com o mundo físico. A maioria dos dispositivos utilizados não são criados pela SmartThings, pois esta tecnologia baseia-se numa integração total do sistema com os dispositivos já existentes, não limitando a nível de tecnologias ou protocolos. É oferecida compatibilidade com os protocolos Z-Wave, Zigbee e Wi-Fi.

- Hub. Este é o ponto central do sistema e garante a comunicação entre os dispositivos, a *cloud* e a aplicação móvel.

- Connectivity Management. É a camada que liga o *hub* e as aplicações móveis aos servidores da SmartThings. Assim sendo, esta camada está dividida em duas partes: Hub Connectivity e Client Connectivity, onde a primeira liga o *hub* à *cloud* e a segunda liga a aplicação móvel à *cloud*.

- Device Type Execution. O sistema determina que tipo de dispositivo é que está ligado, com base nos Device Type Handlers. Desta forma, quando um tipo de dispositivo é selecionado, a mensagem de entrada para a camada Device Type Handlers é filtrada segundo o tipo do dispositivo, e as mensagens de saída passam a ser eventos normalizados no sistema para esse tipo de dispositivo.

- Subscription Management. O propósito desta camada é garantir que os eventos que são transmitidos pela camada Device Type Handlers são coincidentes com tipos de dispositivos

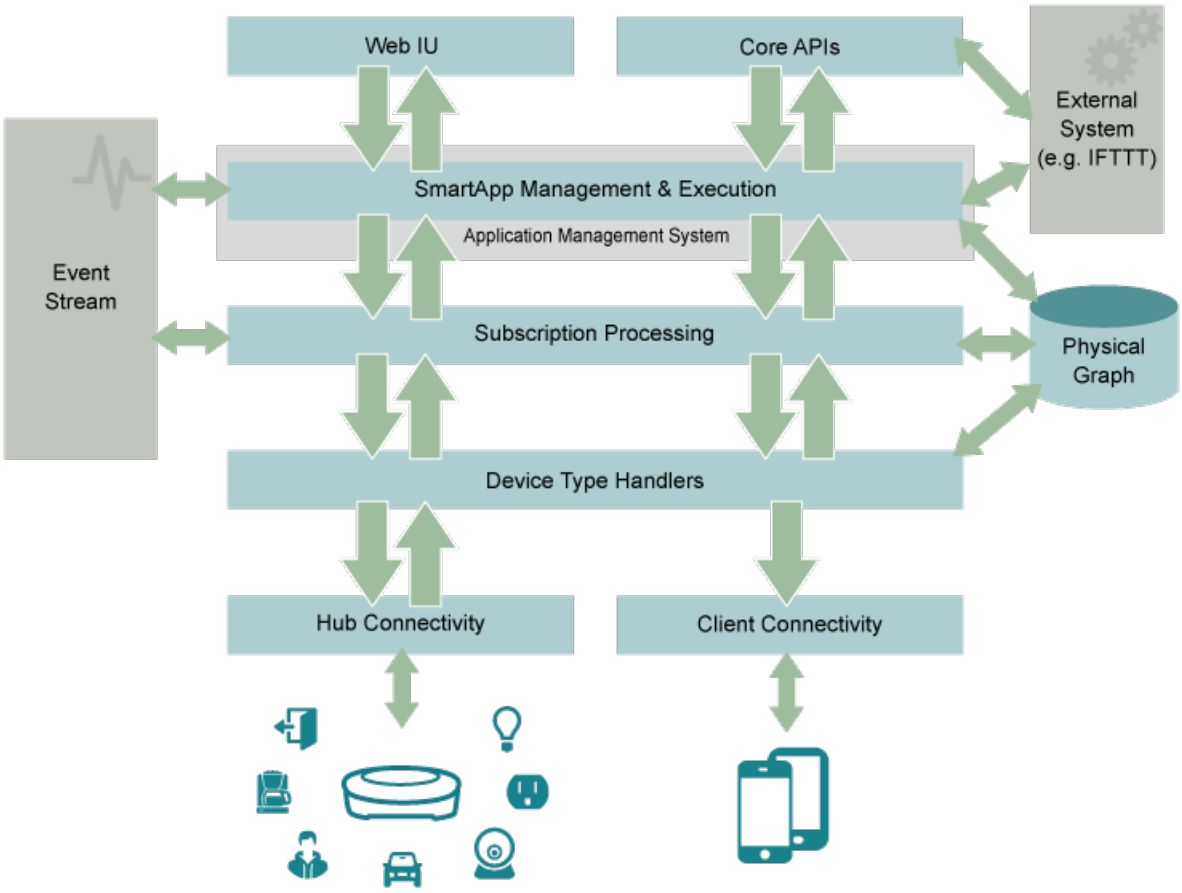

Figura 2.8: Arquitetura do sistema de domótica SmartThings[6]

configurados na SmartApp. A SmartApp tem subscrições que são ativadas quando recebe eventos.

- SmartApp Execution. SmartApp está configurada para funcionar nas suas subscrições (métodos corridos aquando da chegada de eventos, gerados na alteração de estado de um dispositivo), em chamamentos externos (métodos corridos no momento de execução de controlos diretamente na aplicação móvel ou através de *webservice*), ou em métodos calendarizados, ou seja, está normalmente parada, e quando requisitada executa a sua função e pára novamente no fim da tarefa.

- Web UI and IDE. São as partes da arquitetura que permitem monitorizar os dispositivos, *hubs*, localizações e outros aspetos do sistema SmartThings. Funcionam como serviços de *back-office* ao sistema.

2.3 Sistemas de domótica *open-source*

No que toca às ofertas *open-source*, a maioria não representam soluções integradas mas sim *software* capaz de ser implementado para executar funções de controlo domótico. São exemplos de *software open-source* os seguintes:

1. Calaos
2. Domoticz
3. Home Assistant
4. OpenHab

Estas aplicações enumeradas são geralmente integradas com um computador pessoal, um Arduino, ou principalmente com um Raspberry Pi.

Seguidamente vai ser analisada com mais detalhe uma solução nova, chamada myDevices Cayenne. Esta solução vai ser estudada por apresentar algumas semelhanças com o projeto desenvolvido nesta dissertação.

2.3.1 myDevices Cayenne

O myDevices Cayenne é um *software* que permite desenvolver projetos IoT com ações simples do tipo *drag-and-drop*, sendo o sistema completo com o *hardware* Raspberry Pi. Neste sentido, o sistema myDevices está preparado para controlar dispositivos remotamente, ler informações de sensores, memorizar e analisar informação. A figura 2.9 representa um exemplo do painel de instrumentos disponível na aplicação móvel e via web.

O sistema apresenta quatro componentes essenciais:

- Aplicação móvvel. É através da aplicação móvel que é possível configurar e controlar todo o sistema. É altamente personalizável, sendo adicionados novas ações com recurso a um sistema *drag-and-drop*.
- Painel de instrumentos disponível via web. É um meio para configurar e controlar os elementos associados através do acesso pelo navegador da internet.
- *Cloud*. Este elemento é o responsável por guardar informação do utilizador, bem como tratar de despoletar ações, eventos, alarmes e reagir a comandos da aplicação móvel.
- *Agent*. Por último, este elemento trata de estabelecer a comunicação com o servidor.

A aplicação deteta automaticamente todos os Raspberry Pi disponíveis na mesma rede Wi-Fi ou através de Ethernet; ainda assim, depois de configurado é possível utilizar redes móveis para interagir com o sistema, nomeadamente através de um *smartphone*. Para a deteção do Raspberry

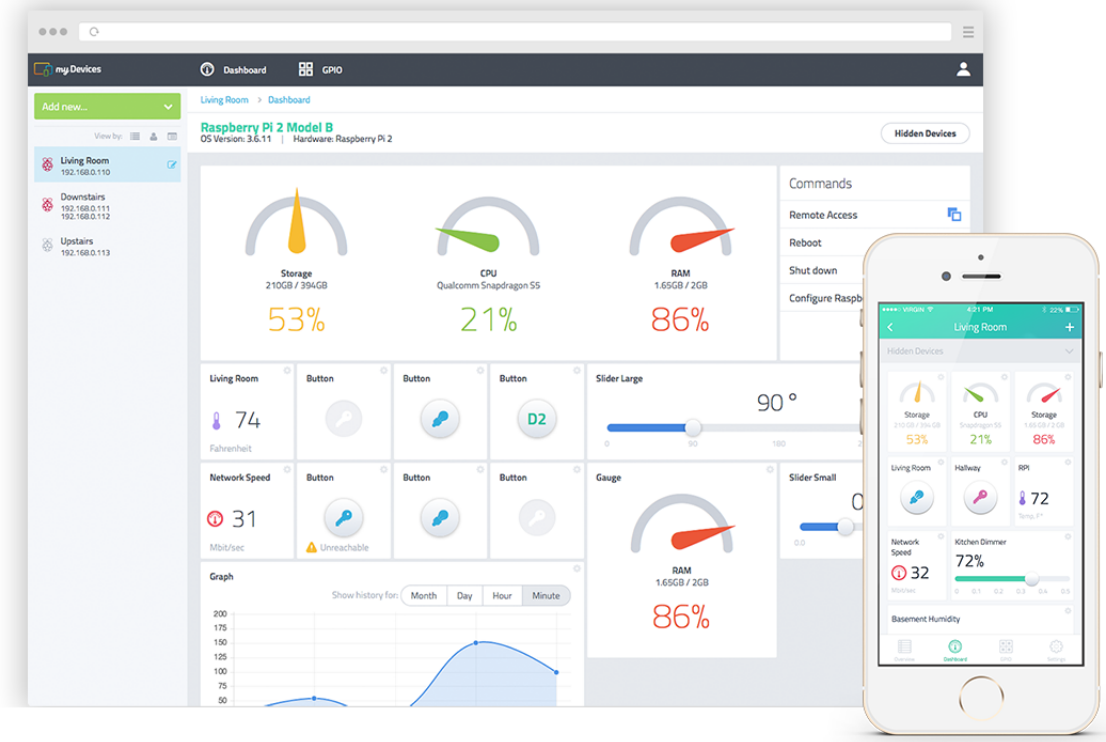

Figura 2.9: Exemplo de dashboard da aplicação móvel myDevices Cayenne[7]

Pi, este deve estar a executar o sistema operativo Raspbian, que é automaticamente configurado pelo serviço myDevices.

Como previamente referido, o sistema funciona com o Raspberry Pi, e permite a adição e controlo sobre atuadores e sensores já pré-definidos geralmente utilizados em conjunto com o Raspberry Pi. Sempre que existe uma alteração de estado, quer provocada pelo utilizador ou automaticamente através de regras, a informação é sempre encaminhada primeiro para a *Cloud* que depois se encarrega de enviar a mesma informação para o outro extremo do sistema, ou seja, se o utilizador decidir alterar o estado de uma luz, essa informação é enviada para a *Cloud* que reeenvia para o Raspberry Pi para atuar devidamente.

2.4 Ambientes de desenvolvimento e multiplataforma

Os dispositivos móveis desempenham cada vez mais um papel fundamental na sociedade. A facilidade que os mesmos trazem na organização e definição de tarefas, calendarização de compromissos, partilha de ficheiros e comunicação dentro das empresas permite o aumento da produtividade e qualidade nos serviços ou produtos fornecidos.

A par da grande evolução existente na melhoria da componente de *hardware*, prevista por Moore e representada segundo a Lei de Moore, os telemóveis e tablets sofreram também nos últimos anos um elevado aperfeiçoamento das suas funcionalidades presentes no seu *software*. Neste sentido, os principais sistemas operativos existentes no mercado são o Android, que lidera a quota de mercado neste contexto, seguido do iOS e posteriormente do Windows 10 Mobile[22].

O sistema operativo Android é baseado em núcleo linux e é propriedade da empresa Google, disponibilizado sob a licença de código aberto, permitindo que entidades terceiras desenvolvam e inovem com ideias próprias. Este sistema operativo foi desenhado para dispositivos com tela sensível ao toque e, hoje em dia, é utilizado em diferentes categorias de dispositivos sendo principalmente conhecido pela sua usabilidade e capacidade de personalização quando executado em smartphones e tablets, contudo é aplicado também em automóveis (Android Auto), relógios (Android Wear) e televisões (Android TV). Esta possibilidade de utilização do mesmo sistema operativo em diferentes categorias de dispositivos, com a liberdade pela qual o Android é conhecida, permite uma integração entre dispositivos e partilha de funcionalidades interessantes e empolgantes.

O iOS é um sistema operativo propriedade da empresa Apple. Ao contrário do que encontramos no sistema operativo da Google, o iOS não é disponibilizado sob licença de código aberto, estando disponível unicamente nos dispositivos desenvolvidos internamente. Assim como o Android, o iOS está presente em diversas categorias de dispositivos: telemóveis (iPhone), leitores multimédia (iPod Touch), tablets (iPad), e televisões (Apple TV). Destaca-se pela sua elevada simplicidade, segurança, e fluídez.

O Windows 10 Mobile é o sistema operativo desenvolvido pela Microsoft. Dos três, é o mais difícil de encontrar no mercado, sendo principalmente associado a telemóveis da empresa Nokia. Assim como o iOS, o Windows 10 Mobile não é disponibilizado sob licença de código aberto. Encontra-se atualmente na versão 10, e começa a apresentar funcionalidades inovadoras na sua integração com o Windows 10 comercializado para computadores, evidenciando a nova funcionalidade chamada *Continuum* - uma inovação que permite um telemóvel comportar-se como um computador, a nível de desempenho e apresentação de informação - que necessita apenas de uma *dock* que liga o telemóvel a um monitor, teclado e rato; apesar de estar a funcionar como um computador, o telemóvel não perde as suas capacidades de comunicação iniciais.

Com o crescimento do mercado dos dispositivos móveis, a necessidade de conseguir desenvolver para todos os sistemas simultâneamente aumentou e, consequentemente, o aparecimento de ambientes de desenvolvimento multiplataformas que facilitem esse mesmo objetivo. As empresas procuram desenvolver aplicações a partir destes ambientes para assim atingirem todo o mercado

de utilizadores de dispositivos móveis de uma forma mais barata e simples. Utilizando estes ambientes de desenvolvimento, as empresas necessitam de ter apenas uma equipa de programadores, capazes de dominar apenas uma linguagem. No sentido de conceber aplicações para dispositivos móveis, os programadores dispõem de ferramentas que vão desde a possibilidade de programar *mobile web apps*, utilizando linguagens de desenvolvimento de aplicações web, até ao desenvolvimento de *native apps*, aplicações concebidas especificamente para o sistema operativo em questão. O seguinte gráfico representa as diferenças entre as diversas abordagens de desenvolvimento de aplicações:

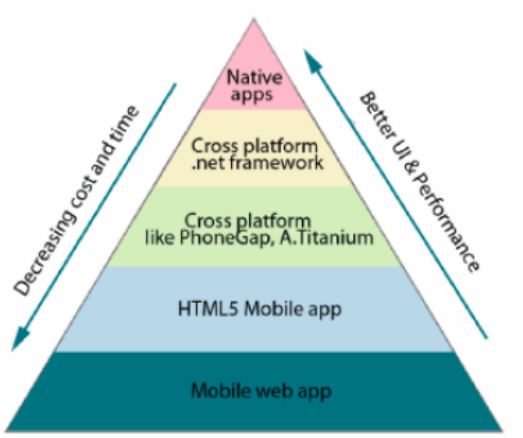

Figura 2.10: Gráfico comparativo com vantagens e desvantagens da utilização de *web apps* e de *native apps*[8]

2.4.1 PhoneGap

O ambiente de desenvolvimento PhoneGap é *open-source* e permite a programação de aplicações para dispositivos móveis recorrendo a HTML5, CSS e JavaScript, possibilitando o desenvolvimento para todos os sistemas operativos.

Começando pela interface, esta é implementada recorrendo às tecnologias web previamente escritas, traduzindo-se numa página de navegador da internet que aproveita a totalidade da largura e altura do ecrã do telemóvel. As páginas da aplicação são ilustradas sem a decoração extra dos navegadores utilizados regularmente, como é o caso do Google Chrome. Contudo, a *WebView*, ou seja, a classe responsável pela interpretação de HTML e CSS e renderização da página, é a mesma que utilizada pelo sistema operativo nativo: no iOS é a classe Objective-C UIWebView e no Android é classe android.webkit.WebView.

O PhoneGap fornece uma API (Application Programming Interface) que permite o acesso a funcionalidades nativas do sistema operativo utilizando JavaScript; a lógica da aplicação é escrita utilizando JavaScript e a API encarrega-se da comunicação com o sistema operativo.

A arquitetura alto-nível das aplicações desenvolvidas através do PhoneGap tem a estrutura representada na figura 2.11.

Figura 2.11: Arquitetura de alto-nível das aplicações PhoneGap[9]

As aplicações funcionam como um cliente com o qual o utilizador pode interagir. O cliente comunica com uma aplicação alojada num servidor, que por sua vez executa a lógica do servidor e obtém os dados provenientes de uma base de dados. As aplicações alojadas em servidores são geralmente escritas em PHP ou .NET, executadas em servidores Apache ou IIS.

Assim sendo, comunicações entre a aplicação e a base de dados não são realizadas diretamente. Geralmente, as comunicações com a aplicação no servidor são efetuadas sobre HTTP, com pedidos em HTML, ou através de serviços como REST-ful XML, JSON ou SOAP. Em suma, as tecnologias utilizadas assemelham-se às vulgarmente implementadas em aplicações web.

A arquitetura do lado do cliente é usualmente uma única página HTML. As informações presentes na página vão alterando conforme são atualizados os elementos, textos e atributos HTML.

Por fim, as aplicações desenvolvidas através do PhoneGap resultam em ficheiros tradicionais para os diferentes sistemas operativos: para iOS a saída é um ficheiro IPA, para Android um ficheiro APK, e para Windows Phone é um ficheiro XAP.

2.4.2 Appcelerator Titanium

A plataforma Appcelerator Titanium é um *framework* disponibilizado em código-aberto que permite a criação de aplicações móveis para os três principais sistemas operativos através de um único código escrito em JavaScript. Os principais componentes do Appcelerator Titanium são o Titanium SDK, que fornece as diferentes APIs, e o Alloy, que permite a criação de interfaces para as aplicações de forma simplificada[23].

O Appcelerator Titanium inclui APIs que permite o acesso às diferentes funcionalidades do sistema operativo, como por exemplo a localização GPS e acelerômetro, bem como componentes gráficos: barras de navegação, menus, caixas de diálogo, entre outros.

A aplicação é traduzida para código nativo de cada sistema operativo; contudo, sendo o JavaScript uma linguagem dinâmica, parte do código não é compilado mas sim interpretado, ou seja, o código escrito em JavaScript é interpretado e invoca uma função nativa para desempenhar a função desejada, codificada em JavaScript pelo desenvolvedor. Por exemplo, ao chamar uma função em JavaScript para a criação de um botão, o código é interpretado e é despoletado um método nativo para criação desse botão. Desta forma, durante o decorrer da aplicação são criados vários objetos *proxy* que servem de pontes de ligação entre as funções nativas (Titanium API) e o código JavaScript.

Os motores JavaScript para interpretar o código são o Mozzila Rhino para o Android e Blackberry, e o JavaScriptCore para o iOS.

O Alloy é um *framework* que permite a criação de aplicações segundo o padrão MVC. Desta forma, através do Alloy é possível criar interfaces através de ficheiros XML, que é bastante comum para os programadores de aplicações móveis.

2.4.3 Xamarin

O Xamarin é uma ferramenta comercial para o desenvolvimento de aplicações móveis. Numa altura em que os 3 grandes possuidores da quota de mercado no mundo dos dispositivos móveis competem intensamente pelo mercado, apresentando inovações e melhorias no seu *software*, o Xamarin apresenta-se como um verdadeiro produto diferenciador no que toca ao desenvolvimento multiplataforma que permite a partilha em média de 75% do código entre plataformas. Ainda assim, ao contrário do que foi descrito sobre o PhoneGap, o Xamarin permite apresentar aplicações nativas que, como foi explicado, se traduz numa elevada melhoria no desempenho e responsividade quando comparadas com aplicações denominadas *web app*.

Para isto, o Xamarin é baseado numa implementação *open-source* a partir do *framework* .NET. Esta implementação inclui o seu próprio compilador C# e principais bibliotecas, também permite aceder a elementos nativos do UI do desenvolvimento de cada plataforma e disponibiliza o seu próprio ambiente de desenvolvimento; contudo, em caso de preferência também é possível utilizar o ambiente da Microsoft para programar, o Visual Studio. Neste sentido, recentemente, o Xamarin foi integrado na nova versão conhecida como Visual Studio 2015 permitindo uma ligação fácil com o serviço Azure da Microsoft e facilitando o armazenamento de dados, autenticação de utilizadores, notificações *push* e sincronização offline.

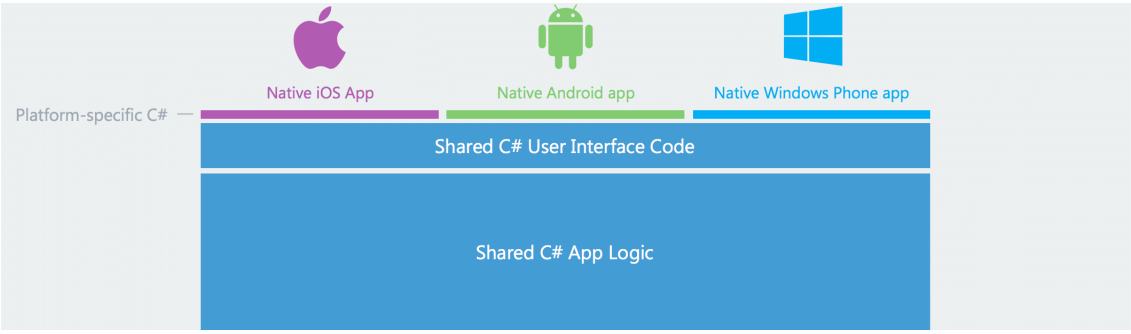

Figura 2.12: Princípio base da plataforma Xamarin[10]

A tecnologia disponibilizada pelo Xamarin é realmente bastante promissora. O número de programadores que adotam esta tecnologia para desenvolver as suas aplicações está constantemente a aumentar, sendo que atualmente mais de 15000 companhias confiam na mesma[24]. Consequentemente, o suporte dado tanto pela sua equipa, como pelos utilizadores desta tecnologia, é muito positivo e esta encontra-se em permanente melhoria e atualização.

2.4.3.1 Compilação e acesso a SDKs

O Xamarin permite a utilização de código para desenvolvimento de aplicações para as três diferentes plataformas; no entanto, a implementação em cada um dos sistemas é bastante diferente[10].

No que diz respeito ao iOS, o código é compilado na totalidade antes de qualquer utilização; este conceito é conhecido como compilação *ahead-of-time*. O código é compilado para a linguagem de compilação ARM. A *framework* .NET é incluída eliminando todas as classes que não são utilizadas durante um processo chamado *linking*, que é comum ao iOS e Android e por isso é explicado posteriormente. O Xamarin.iOS disponibiliza o *framework* CocoaTouch SDK da Apple que pode ser referenciado no código em C# e contém todos os controlos para interface com o utilizador. Este sistema operativo não permite a geração de código durante a execução, limitando o desenvolvimento em algumas funcionalidades, entre as quais o *Reverse Callback*.

Relativamente ao sistema operativo Android, o código é compilado para a linguagem IL (ou CIL, Common Intermediate Language) e é empacotado com o *MonoVM (Virtual Machine)* mais *JIT (Just-In-Time) compilation*; este conceito permite que as aplicações compilem o código durante a sua execução, ao contrário do que acontece para o iOS. O Xamarin.Android disponibiliza o Android SDK da Google, facilitando o acesso aos controlos de interface com o utilizador.

Finalmente, para o Windows 10 Mobile, o programa é compilado para a linguagem IL e não é necessário o uso de ferramentas do Xamarin; ainda assim, recorrer ao Xamarin para desenvolver as aplicações possibilita a utilização do código elaborado para o iOS ou Android. Como o Windows 10 Mobile não faz parte da plataforma Xamarin, aquando do desenvolvimento das aplicações em C# o SDK está implicitamente disponível incluindo os controlos do Silverlight/XAML de interface com o utilizador.

O *linking*, falado anteriormente, é uma ferramenta fornecida pelo Xamarin que permite reduzir acentuadamente o tamanho da aplicação (em cerca de 25%); em contrapartida, o tempo de compilação é mais elevado. Para isto, é efetuado um processo que verifica todas as bibliotecas que podem ser necessárias durante a execução do programa, garantindo a maior das certezas para evitar maus funcionamentos ou reações indesejadas, e elimina aquelas que não são definidas como necessárias.

2.4.3.2 Desenvolvimento e limitações

Para desenvolver aplicações para o iOS é necessária a utilização de um computador Mac. Os programas podem ser escritos utilizando o *plugin* do Xamarin para o Visual Studio, mas um computador Mac é necessário para a compilação e licenciamento do programa. O IDE Xcode da Apple tem de estar obrigatoriamente instalado para fornecer o compilador e o simulador para testes.

O desenvolvimento de aplicações para Android necessita que estejam instalados os SDKs Android e Java; estes fornecem o compilador, emulador e outras ferramentas requisitadas para compilação, implementação e teste. Neste sentido, o Xamarin disponibiliza um instalador único

que configura o sistema com os pré-requisitos mencionados. Para testar e implementar o programa num dispositivo não é precisa qualquer licença por parte da Google.

Por fim, para o Windows 10 Mobile, os programas são desenvolvidos e compilados diretamente através do Visual Studio.

A seguinte tabela apresenta as diferentes possibilidades de desenvolvimento dos programas para as 3 plataformas:

Development OS	Mac OS X	Windows	
IDE	Xamarin Studio	Xamarin Studio	Visual Studio
iOS	Y	-	Y
Android	Y	Y	Y
Windows Phone	-	-	Y

Tabela 2.2: Tabela com as diferentes possibilidades de desenvolvimento através do Xamarin para os três sistemas operativos[10]

2.4.3.3 Interfaces gráficas

O grande benefício de utilizar o Xamarin para desenvolver a interface gráfica para a aplicação prende-se com a capacidade de aproveitamento dos controlos nativos de cada sistema operativo, criando aplicações da mesma forma que em Objective-C e Java (para iOS e Android respetivamente)[15].

As interfaces podem ser concebidas recorrendo a código ou através de ferramentas de desenho preparadas, com o método *drag-and-drop*. A vantagem de desenvolver as intefaces através de código é permitir às mesmas que tenham um comportamento dinâmico nos ecrãs dos dispositivos, nomeadamente, entre telemóveis e tablets que apresentam tamanhos de ecrã diferentes. Por outro lado, desenhar a partir das ferramentas fornecidas é menos demoroso e complicado, sendo que estas fornecem inclusivamente acesso às propriedades dos elementos; desta forma, os ficheiros ficam acessíveis no ambiente de desenvolvimento para iOS no formato .STORYBOARD, para Android no formato .AXML e para Windows 10 Mobile no formato .XAML.

Para além do mencionado previamente, outra vantagem da utilização das ferramentas para desenhar as interfaces prende-se com a similaridade dos objetos existentes entre as 3 plataformas. Por outro lado, existem elementos diferentes para cada sistema operativo, como por exemplo:

- iOS: navegação hierárquica através de um simples botão de regresso, abas no extremo inferior do ecrã;
- Android: botão de regresso integrado no *software* ou *hardware* do sistema, menu de ação, abas no extremo superior do ecrã;
- Windows 10 Mobile: botão de regresso no *hardware* do sistema, visualização através de panoramas com divisões animadas.

Neste sentido, e principalmente direcionado para o iOS e o Android, existem diferentes pacotes de APIs que fornecem ferramentas específicas: Xamarin.iOS, Xamarin.Android e Xamarin.Forms. O Xamarin.Forms é ideal para a utilização em aplicações de inserção de dados, protótipos e provas de conceito, aplicações que necessitam de funcionalidades partilhadas; ou seja, em suma para aplicações onde o importante é a partilha de código e não o aspeto gráfico. Os Xamarin.iOS e Xamarin.Android são preferíveis em aplicações que usem interações específicas do sistema operativo, aplicações com uma interface gráficas bastante cuidada e intuitiva; concluindo, são ideais para utilizar em aplicações onde a interface e a aparência são mais importantes que a partilha de código entre plataformas.

2.4.3.4 Shared Projects

Este método engloba-se no grupo de métodos oferecidos pelo Xamarin para partilha de código, composto por: *Shared Projects* e *Portable Class Libraries*, que irá ser falado de seguida[25].

Este método é considerado o mais simples de utilizar. É baseado no seguinte princípio: o código é dividido por pastas, sendo que cada pasta contém um tipo de código, de entre os seguintes: código comum a todas as plataformas, código destinado ao Android, código destinado ao iOS e, por fim, código destinado ao Windows 10 Mobile. Assim sendo, no código destinado a todas as plataformas, os caminhos são definidos por condições do tipo *if*, tendo como objeto de análise o sistema operativo em funcionamento. Este conceito é facilmente compreendido analisando a seguinte figura:

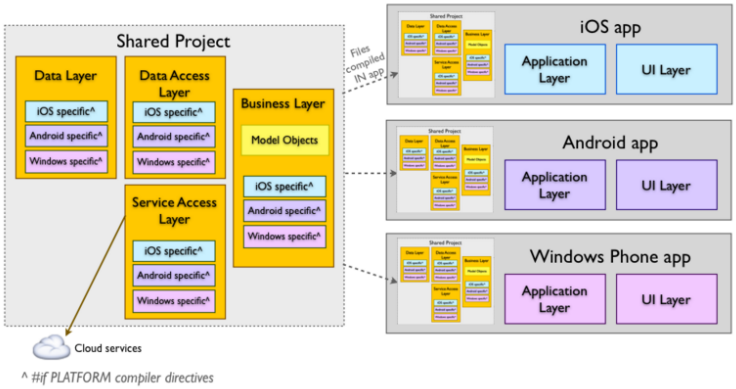

Figura 2.13: Lógica de funcionamento do método *Shared Projects* [11]

Os principais benefícios deste método são: permite a partilha de código entre as plataformas, o código partilhado pode ser dividido em ramos para que cada plataforma rentabilizada ao máximo e o código permite incluir referências a ficheiros específicos de cada plataforma. Posto isto, a principal desvantagem está relacionada com a não criação de um executável independente (DLL) para o programa, mas sim para cada biblioteca utilizada/referenciada. Resumindo, podemos definir o método *shared projects* como um conjunto de bibliotecas que quando agrupadas para uma determinada plataforma produzem uma aplicação.

2.4.3.5 Portable Class Libraries

Neste método de partilha de código, cada DLL criado para o programa desenvolvido está destinado a uma plataforma específica. Isto previne que um programa desenvolvido para Windows 10 Mobile seja reutilizado no Xamarin.iOS ou Xamarin.Android.

Apesar disto, aquando da criação de uma *Portable Class Library*, é possível especificar mais que uma plataforma sobre a qual é desejado que o programa execute. Estas escolhas são gravadas num identificador de perfil que descreve as plataformas que a livraria suporta. A seguinte tabela representa as possibilidades de integração entre diversas plataformas:

Feature	.NET Framework	Windows Store Apps	Silverlight	Windows Phone	Xamarin
Core	Y	Y	Y	Y	Y
LINQ	Y	Y	Y	Y	Y
IQueryable	Y	Y	Y	7.5 +	Y
Serialization	Y	Y	Y	Y	Y
Data Annotations	4.0.3 +	Y	Y		Y

Tabela 2.3: Possibilidades de integração de *frameworks* em diferentes sistemas operativos[11]

Na tabela disponibilizada é possível confirmar versatilidade oferecida pelo Xamarin.iOS e Xamarin.Android juntamente com o Visual Studio.

A figura mostrada de seguida representa o funcionamento descrito anteriormente para este método:

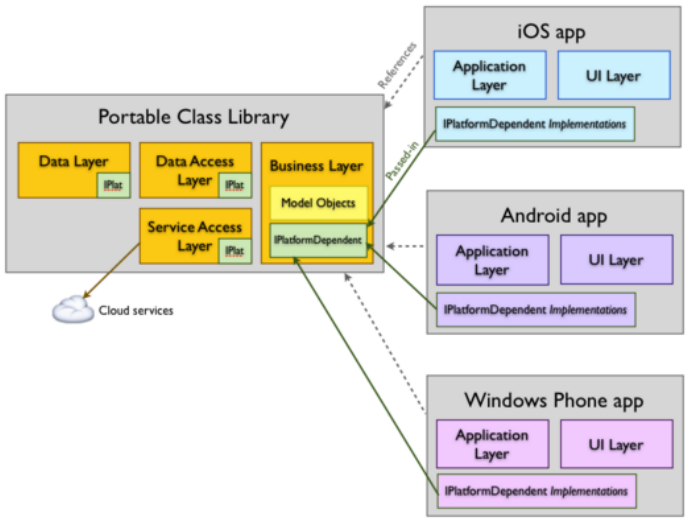

Figura 2.14: Lógica de funcionamento do método *Protable Class Libraries*[11]

Os principais benefícios deste método passam pela partilha total de programas desenvolvidos, sendo possível a referenciação por parte de outra aplicação para o programa inicial. Contudo, este

método apresenta a desvantagem de incompatibilidade de classes e livrarias entre as diferentes plataformas; algumas destas adversidades podem ser contrariadas recorrendo ao método *Dependecy Injection*[11].

2.5 Conclusão

Atualmente na área da automação residencial, podemos verificar que existem muitos produtos emergentes no mercado, apresentando diferentes propostas. Os mecanismos de controlo oferecidos por cada companhia são aproximadamente os mesmos. Contudo, o modelo de negócio e as propostas de valor tendem a diferenciar-se, essencialmente, entre sistemas *open-source* e sistemas fechados. A principal adversidade à compra de estes sistemas continua a ser o custo, sobretudo no que diz respeito à componente de *hardware* que não compensa o ganho inerente à instalação de um sistema de domótica.

No que diz respeito aos sistemas fechados encontramos maioritariamente conjuntos integrados de *software* e *hardware*, criando assim um ambiente próprio para o funcionamento dos seus serviços e que oferecem fiabilidade e extensibilidade nos seus sistemas. Os protocolos utilizados são variados, apesar de maioritariamente incidirem no Z-Wave, Zigbee e Wi-Fi. Contudo, começam a aparecer soluções que integram meios de *hardware* de outros fabricantes, como é o caso do SmartThings que permite a utilização de dispositivos WeMo.

Por outro lado, os sistemas *open-source* são normalmente meios de *software* para programação e criação dos próprios sistemas de domótica. Oferecem soluções preparadas para criação de regras, adição de dispositivos, entre outras funcionalidades, e obrigam à compra de *hardware* onde instalar a aplicação de controlo, como é o caso do Raspberry Pi, que é uma solução muito procurada.

Capítulo 3

Análise de Requisitos e Arquitetura da Solução

Este capítulo contém, numa primeira instância, o levantamento dos requisitos que foram tomados como base para desenvolvimento do projeto, partindo dos casos de uso. Num segundo momento, será apresentada a arquitetura adotada para o sistema, a par da especificação de algumas funcionalidades.

3.1 Casos de uso

Os diagramas de caso de uso pertencem à Unified Modeling Language (UML). A UML é uma linguagem de modelação, utilizada na área de engenharia de *software*, e distingue-se por ser uma ferramenta que permite obter uma interpretação gráfica do sistema, pondo em evidência as suas questões de semântica e relacionais, mantendo a abstração no alto nível. Para além disto, a UML representa ainda uma forma normalizada de projetar o sistema. Particularmente, os casos de uso são o método ideal para descrever a interação entre o sistema e os seus atores.

Neste projeto em questão, foi admitido que toda a execução de tarefas relativas a *back office* estão presentes num computador externo ao sistema aqui considerado. Assim sendo, o ator presente neste sistema é apenas o utilizador da aplicação móvel.

Os modelos de caso de uso a ser apresentados pertencem a dois pacotes distintos: Autenticação e Funcionamento. O primeiro pacote representa os casos de uso antes do utilizador efetuar o seu login, ao invés do segundo pacote que representa os casos de uso após uma autenticação bem sucedida.

É importante referir que o modelo de casos de uso representa o sistema de um nível muito alto, pelo que cada caso indicado pode ser suscetível de divisão em mais pequenos casos.

3.1.1 Casos de uso: pacote Funcionamento

A seguinte figura representa os casos de uso relativos ao pacote Funcionamento.

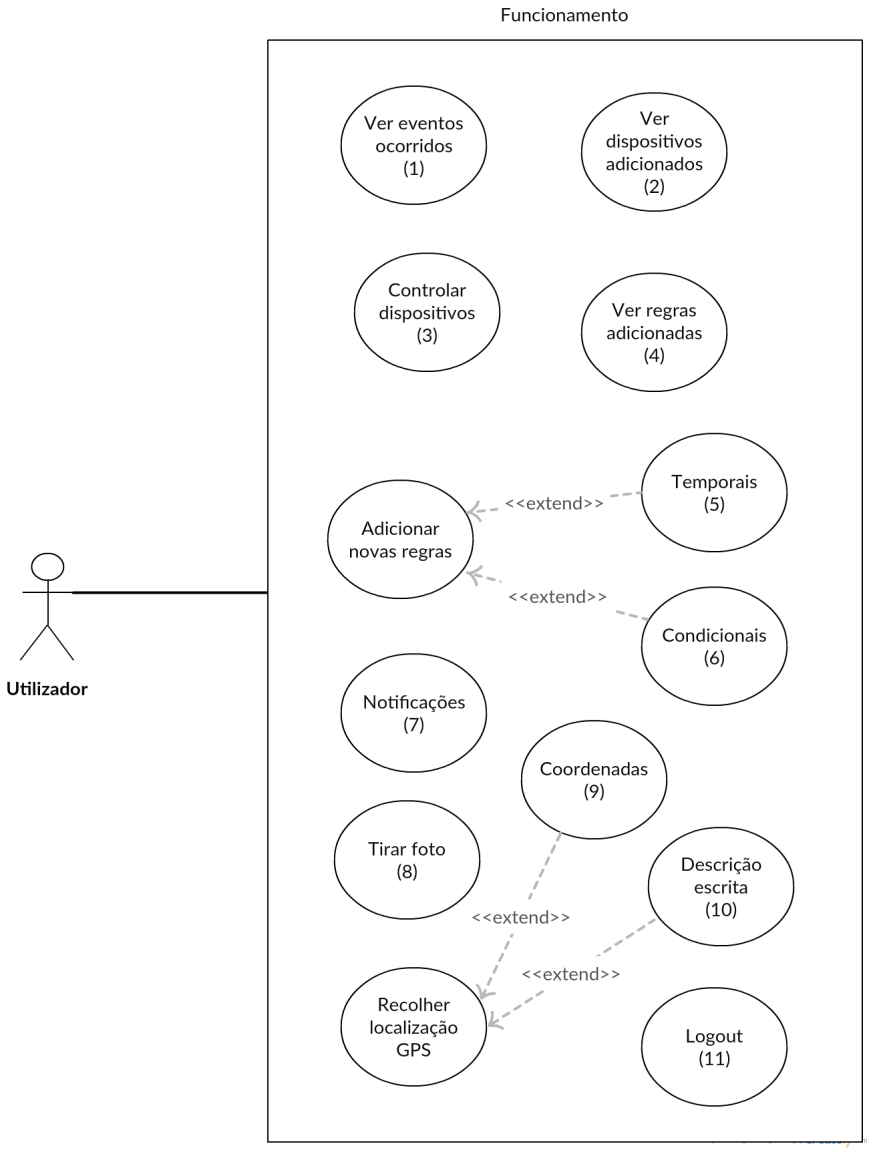

Figura 3.1: Modelo de casos de uso do pacote Funcionamento

Os casos de uso são utilizados neste documento para apresentar as funcionalidades principais do sistema do ponto de vista dos utilizadores, sendo estes pormenorizados através da sua relação com os requisitos presentes no capítulo seguinte, através da refernciação numérica entre os casos de uso das figuras 3.1 e 3.2 com os requisitos das tabelas 3.1 e 3.2.

3.1.2 Casos de uso: pacote Autenticação

A seguinte figura mostra os casos de uso relativos ao pacote Autenticação.

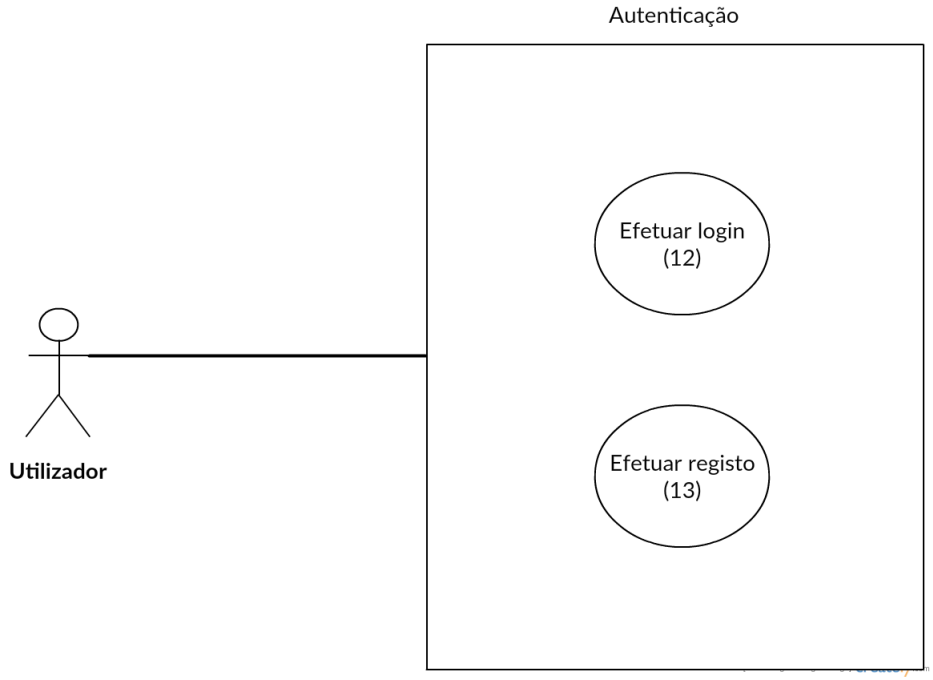

Figura 3.2: Modelo de casos de uso do pacote Autenticação

3.2 Requisitos funcionais

Os requisitos funcionais de um sistema traduzem as funções que o sistema terá de desempenhar e disponibilizar aos seus utilizadores. Neste sentido, as funções podem ser interpretadas como processos que recebem uma entrada, tratam-na e apresentam uma saída. Na tabela 3.1 estão apresentados os requisitos bem como o seu plano de teste para a implementação.

ID	**Descrição**	**Plano de teste**	**Casos de uso**
RF01	O sistema deverá ter associado um processo de autenticação com recurso a nome de utilizador e palavra-chave.	Introduzir os dados do utilizador corretamente e incorretamente em tentativas seguidas e verificar os resultados obtidos em ambas.	11, 12 e 13
RF02	A aplicação deverá fornecer ao utilizador informação em tempo real relativa ao estado dos diferentes aparelhos conectados ao sistema.	Aceder à área de estado dos dispositivos e verificar que todos os dispositivos estão descritos, inclusive o seu estado atual.	2

RF03	A aplicação móvel deverá permitir ao utilizador ligar e desligar aparelhos elétricos e/ou eletrónicos remotamente, com o tempo de resposta menor ou igual a 1 segundo.	Clicar no botão disponível para alterar o estado de um dispositivo na aplicação móvel.	3
RF04	O utilizador deve ter acesso a um histórico de eventos, filtrado por dispositivo associado ao evento.	Aceder à página Dashboard na aplicação e visualizar os eventos com informações relativas como data e dispositivo associado.	1
RF05	O utilizador deve ser capaz de visualizar todas as regras definidas: condicionais e temporais.	Aceder à área de visualização de regras e verificar todas as regras definidas e ativas no momento.	4
RF06	A aplicação deve permitir ao utilizador a programação de atividades, ou seja, ativação ou desativação de aparelhos com base no dia e hora.	Aceder à área de definição de regras temporais e definir um horário e estado para um determinado dispositivo.	6
RF07	O utilizador deverá ter a possibilidade de definir regras condicionais para controlo dos dispositivos.	Aceder à área de definição de regras condicionais e atribuir uma condição para alteração do estado de um dispositivo.	5
RF08	O sistema deverá ter uma camada de acesso ao *hardware*. O Raspberry Pi deverá agir sobre os dispositivos segundo as regras previamente definidas pelos utilizadores. O sistema deve ser capaz de funcionar com a presença de pelo menos 100 regras guardadas na base de dados.	Após a definição de regras, verificar que o Raspberry Pi altera o estado do dispositivo em conformidade com as mesmas.	4, 5 e 6
RF09	Dependendo do tipo de evento registado, a aplicação móvel deve fornecer ao utilizador notificações sobre o mesmo.	Provocar a alteração de um aparelho e verificar que é automaticamente gerada uma notificação na aplicação móvel.	7
RF10	As notificações fornecidas ao utilizador pela aplicação móvel devem ser sinalizadas com cores diferentes segundo o sucesso ou insucesso da acção.	Provocar o sucesso e insucesso em determinadas tarefas e verificar que no dispositivo móvel as notificações aparecem com cor relativa ao sucesso e insucesso.	7
RF11	A aplicação deve disponibilizar uma página capaz de utilizar os recursos do dispositivo móvel para tirar uma fotografia e guardá-la na base de dados, com uma descrição associada.	Clicar no botão disponível para ver a foto e com recurso a uma função que recolha a foto da base de dados verificar que a foto está visível para posterior uso.	8

RF12	A aplicação na mesma página disponível para tirar a foto deve disponibilizar uma botão para recolher a localização atual do utilizador através do recetor GPS.	Clicar no botão disponível para obter a localização GPS e verificar que a localização indicada corresponde à localização atual do utilizador.	9 e 10

Tabela 3.1: Requisitos funcionais do sistema

Nesta lista de requisitos é de salientar que os requisitos RF11 e RF12 foram adicionados numa fase posterior, com o projeto em desenvolvimento.

3.3 Requisitos não-funcionais

Os requisitos não-funcionais são os requisitos que estão relacionados com o uso da aplicação no que diz respeito à sua usabilidade, desempenho, segurança, interface. São atributos do produto que não representam funcionalidades mas fornecem um acréscimo à sua qualidade.

ID	**Descrição**	**Plano de teste**
RNF01	O sistema físico deverá ser acedido através da aplicação móvel a partir de qualquer local com ligação à internet.	Executar a aplicação na mesma rede que o Raspberry Pi e a partir de outra distinta e verificar que a conexão é estabelecida em ambos os casos.
RNF02	O serviço disponibilizado pelo Raspberry Pi deverá ser acessível por vários dispositivos em simultâneo.	Conectar dois smartphones em simultâneo ao Raspberry Pi e realizar pedidos diferentes partindo de aparelhos diferentes.
RNF03	Em caso de falhas na rede, o sistema continuará a funcionar, nomeadamente a camada de acesso a hardware.	Provocar uma falha de rede no Raspberry Pi e verificar que os componentes de *hardware* apresentam resultados segundo as regras previamente guardadas.
RNF04	A aplicação deverá ser compatível com sistemas operativos Android 4.0 e superiores, e sistemas operativos iOS7 e superiores.	Ao executar a aplicação desenvolvida em smartphones com os sistemas operativos referidos a aplicação deverá funcionar disponibilizar todas as funcionalidades desenvolvidas.
RNF05	Todos os componentes de hardware a controlar deverão ser acessíveis pela aplicação assim como através de interruptores ou outros meios físicos.	Executar um comando através da aplicação, seguido de uma alteração direta através de um interruptor, e verificar a alteração do estado do componente em ambas as situações.

RNF06	A par da camada de acesso a hardware, a camada de acesso a base de dados deve estar também disponibilizada pelo Raspberry Pi.	Desligar o Raspberry Pi e tentar efetuar uma autenticação na aplicação móvel; verificar que a autenticação é mal sucedida.

Tabela 3.2: Requisitos não-funcionais do sistema

3.4 Arquitetura do sistema

Finda a análise de requisitos, é fundamental projetar uma arquitetura para o sistema Mordomo.

Considerando o contexto no qual a empresa lança este projeto, o sistema Mordomo é projetado como sendo um teste piloto para prova de vários conceitos. Em primeiro lugar e o ponto mais relevante, o início no desenvolvimento de aplicações móveis, incorporando a utilização de algumas funcionalidades nativas dos *smartphones*, como é o caso da localização GPS e utilização da câmara, podendo estender-se ao uso de sensores como o acelerômetro ou sensor de luminosidade. Em segundo lugar, a utilização do Raspberry Pi para execução de programas com algum nível de complexidade, aproveitando a sua interface GPIO. Desta forma, o sistema Mordomo foi pensado para controlar e monitorizar dispositivos básicos, nomeadamente componentes elétricos presentes habitualmente num espaço residencial e, por isso, apesar de a aplicação móvel ser o ponto que requer maior atenção e detalhe, é fundamental considerar na análise da arquitetura o Raspberry Pi que permite interpretar sinais elétricos enquanto entradas e gerar igualmente saídas. Por último, o sistema deve ainda ser capaz de armazenar informação para ter uma gestão autónoma. Assim sendo, o sistema está dividido em duas partes essenciais: a aplicação móvel e o Raspberry Pi.

3.4.1 Arquitetura do *hardware*

Este capítulo destina-se ao estudo da ligação entre os diferentes componentes físicos que incorporam o sistema Mordomo. A figura 3.3 ilustra a montagem efetuada.

Imprescindível para o sucesso das comunicações entre os diferentes componentes do sistema é a ligação das duas partes do sistema à internet. O Raspberry Pi utilizado foi a última versão pelo que integra uma placa de rede com acesso à rede wi-fi, para além da entrada ethernet existente em qualquer versão. Da mesma forma, todos os smartphones possuem capacidade de ligação a redes wi-fi ou redes móveis como 3G ou 4G. Assim sendo, este requisito está integralmente garantido pelas especificações dos componentes constituintes do sistema.

Como é possível concluir a partir da observação do esquema representado, os utilizadores conseguem interagir com o sistema Mordomo apenas através da aplicação móvel, e vice-versa. Isto significa que todas as adições e definições de dispositivos, controlos em tempo real, parametrizações e definições de regras são feitos através da aplicação, da mesma forma que todas as transmissões de informação aos utilizadores, nomeadamente as notificações e registo de eventos.

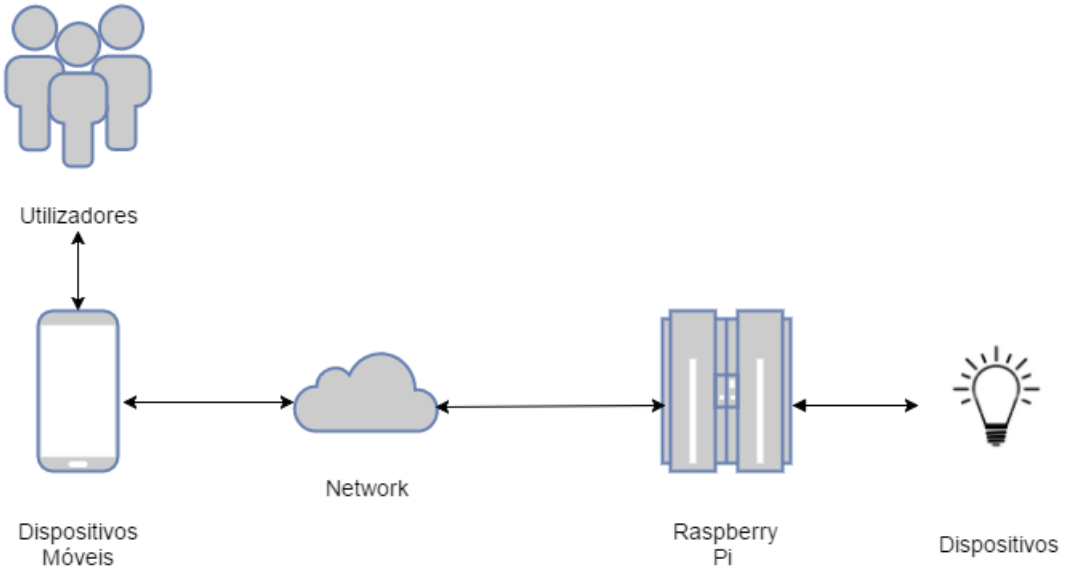

Figura 3.3: Esquema da arquitetura do *hardware* do sistema

No outro extremo do sistema físico temos o Raspberry Pi. Este componente foi escolhido pois é um mini-computador, capaz de executar programas complexos, e ao mesmo tempo apresentar uma interface de entradas e saídas digitais (GPIO). O Raspberry Pi está responsável por cinco constituintes da aplicação do sistema:

- Controlo autónomo.
- *Web Service*.
- Acesso a base de dados.
- Acesso à inteface GPIO.
- Base de dados.

3.4.2 Arquitetura lógica

O *software* desenvolvido para o sistema Mordomo apresenta várias partes, disponibilizadas nos componentes *hardware* previamente referidos. Neste capítulo serão estudadas essas partes e as comunicações presentes entre elas.

A figura 3.4 apresenta um esquema do *software* implementado.

Como é possível verificar, a aplicação móvel apenas comunica com o *web service*. Nesta arquitetura, o *web service* é o centro do *software*. É o *web service* que liga todas as partes da aplicação, e é a aplicação responsável por garantir a lógica da comunicação entre a base de dados, a aplicação móvel, a aplicação de controlo autónomo, e a interface GPIO. O *web service* está ligado a três diferentes programas:

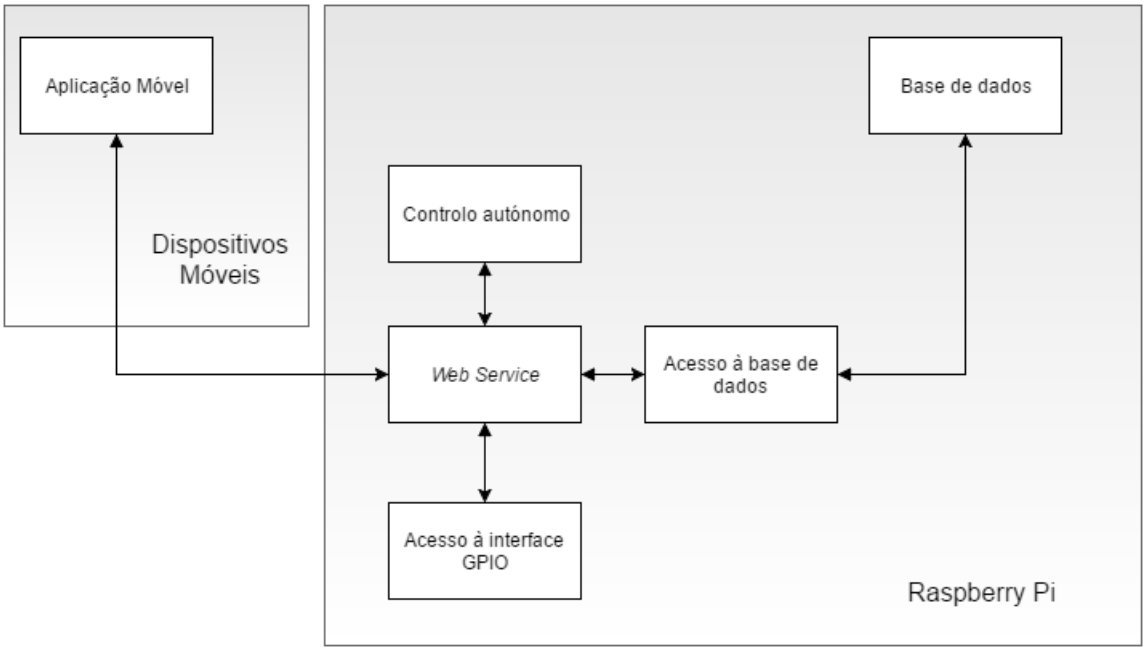

Figura 3.4: Esquema da arquitetura da lógica da aplicação do sistema

- Controlo autónomo. Esta é uma aplicação independente que gere a execução das regras criadas pelo utilizador.

- Acesso a base de dados. É uma parte da aplicação que executa as funções de comunicação com a base de dados, estabelecendo a comunicação e realizando as *queries*.

- Acesso à inteface GPIO. Divisão da aplicação que estabelece ligação com a interface GPIO do Raspberry Pi.

Por último, a divisão do *software* de acesso à base de dados é a única que comunica diretamente com a base de dados, a qual é responsável por armazenar toda a informação relevante para o sistema e está presente na memória interna do Raspberry Pi.

As tecnologias necessárias para implementar todas as aplicações vão ser descritas posteriormente neste documento, bem como o modelo da base de dados.

Dada a importância da aplicação neste sistema, e sendo este projeto apenas uma primeira fase do que poderá vir a ser no futuro, é fundamental que a arquitetura da aplicação pensada possibilite uma elevada taxa de crescimento e melhoria em posteriores intervenções. Com este objetivo em mente, a arquitetura adotada no desenvolvimento da aplicação assenta sobre os conceitos da arquitetura a três camadas, ou *design pattern* MVC (Model-View-Controller).

A arquitetura a três camadas tem como base a divisão do *software* da seguinte forma:

- Camada de dados (Model). Esta camada é responsável pelo envio e receção de dados, estabelecendo a comunicação com a base de dados em serviço. Assim sendo, mantém contacto com a camada da lógica do negócio para interpretação dos pedidos e envio dos dados.

- Camada da lógica de negócio ou acesso a dados (Controller). Esta camada é a que controla toda a aplicação, englobando todos os processos inerentes aos procedimentos oferecidos pela aplicação, a fim do sistema poder representar resultados ao utilizador. Esta camada é ainda responsável por estabelecer a comunicação entre a camada de apresentação e a camada de dados.

- Camada de apresentação (View). Esta camada está encarregue de apresentar informação ao utilizador de forma compreensível e oferecer meios para desencadear a execução de tarefas no sistema.

A figura 3.5 pretende ilustrar um exemplo para compreensão da arquitetura descrita. Como é possível verificar no exemplo representado, a camada de apresentação disponibiliza um meio para iniciar um procedimento chamado "get total sales". Posteriormente, esta informação é enviada para a camada de lógica que executa o pedido efetuado e procura obter informação da base de dados através da questão realizada à camada de dados. A camada de dados por sua vez comunica com a base de dados, ou sistema de armazenamento, e inicia-se o fluxo no sentido oposto.

A camada de dados envia a informação obtida para a camada de lógica; esta por sua vez trata a informação obtida, que neste caso em concreto é adicionar todas as vendas e, por fim, reencaminha este valor para a camada de apresentação. A camada de apresentação apenas recebe o valor e apresenta-o ao utilizador.

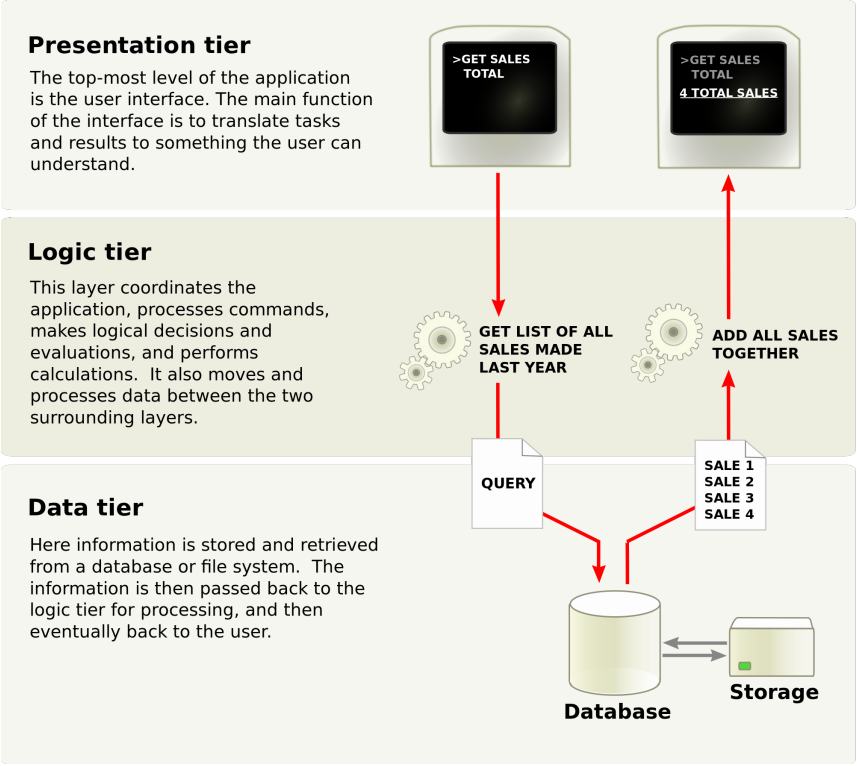

Figura 3.5: Arquitetura a três camadas[12]

Esta arquitetura tem como principal vantagem o desenvolvimento dos diferentes módulos separadamente, possibilitando a reutilização de código e a melhoria separada de cada um deles, ou seja, a apresentação da aplicação pode ser melhorada posteriormente sem que isso interfira com a sua lógica e acesso a dados.

Neste projeto procurou-se adaptar a aplicação ao *design pattern* MVC como ilustra o esquema da figura 3.6.

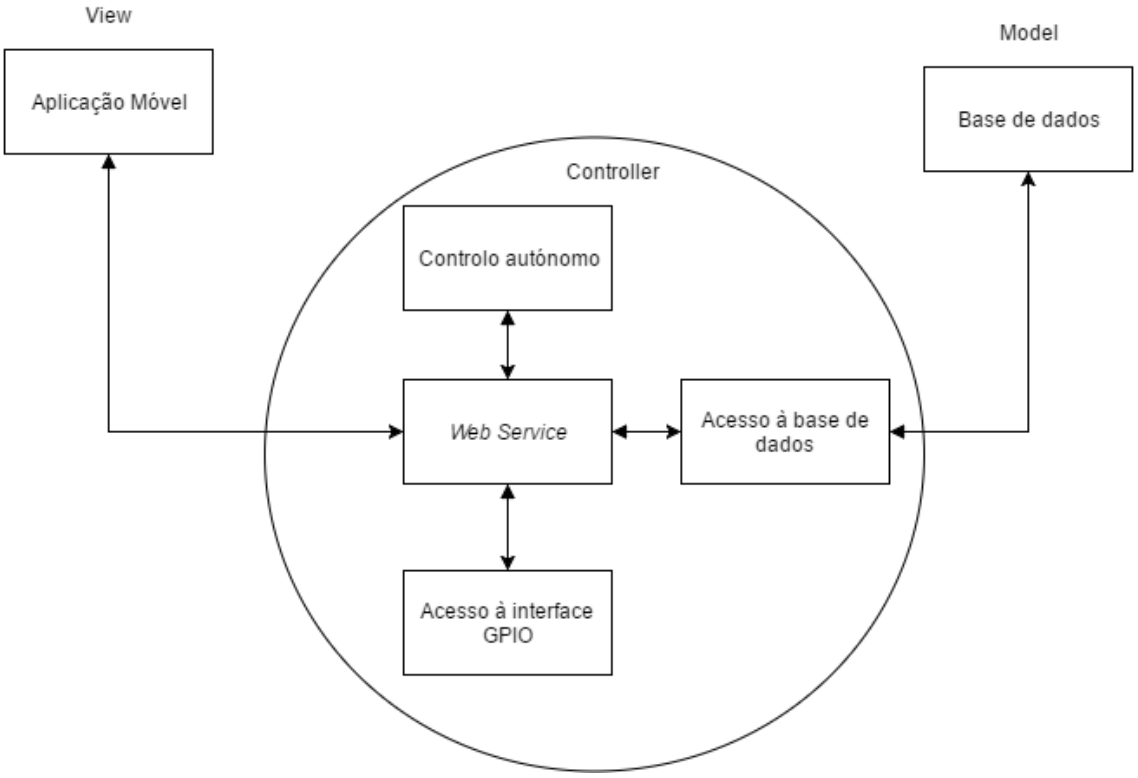

Figura 3.6: MVC aplicado ao sistema Mordomo

3.5 Arquitetura da aplicação

No subcapítulo anterior foi explicada a arquitetura do sistema pensada para satisfazer os requisitos exigidos neste projeto. Neste subcapítulo será abordada a arquitetura da aplicação, enquanto *software* constituído por programas distintos que comunicam entre si e se completam. À vista disto, cada programa será explicado individualmente, fazendo referência à sua comunicação com os demais. As divisões de acesso a base de dados e acesso à interface GPIO não serão analisadas porque apenas incorporam funções de acesso sem arquitetura relevante.

3.5.1 Aplicação móvel

A aplicação móvel é a parte do projeto que disponibiliza funções para interagir com o utilizador, bem como a interface responsável para isso. Serão abordadas duas questões neste capítulo: o

diagrama de classes do *software* da aplicação, e o esquema da interface desenvolvida.

3.5.1.1 Diagrama de classes

Os diagramas de classes, tal como os de casos de uso, pertencem à Unified Modelling Language (UML).

Os diagramas de classes têm como objetivo apresentar graficamente todas as classes do sistema e relações entre as mesmas, que servem de modelo para os objetos a serem criados. É uma ferramenta muito útil para modelação do *software*, pois representa como o mesmo deve ser desenvolvido sem limitar a sua implementação, visto que não restringe quanto às tecnologias a utilizar. As classes são descritivos dos objetos do sistema que apresentam os mesmos atributos, métodos, relações e comportamento. As classes podem ser ainda generalizações de outras classes, sendo isso representado no diagrama com as setas correspondentes.

A figura 3.7 ilustra o diagrama de classes elaborado para ser tomado como ponto de partida para o desenvolvimento da aplicação móvel.

No diagrama é possível observar que existe uma classe denominada App, que é a classe na qual o programa se inicia e que disponibiliza essencialmente as funções necessárias para mostrar as páginas base da aplicação móvel. Para além destas, a classe App é responsável por disponibilizar as funções relativas ao registo de novos utilizadores no sistema, e de guardar a informação sobre a autenticação do utilizador atual.

Seguidamente, verifica-se que a classe App está conectada a uma classe chamada User. Esta nova classe representa os dados do utilizador na aplicação e, sendo que cada aplicação apenas se encontra com um utilizador ligado num determinado momento, a conexão entre as duas classes é de um para um. A classe User disponibiliza para a aplicação os atributos relativos ao utilizador autenticado, e as funções para autenticação e saída da aplicação.

Associadas à classe App encontram-se mais duas: Occurrences e Devices. A primeira, existe para executar funções e guardar informação sobre as fotos retiradas com junção da localização GPS; a segunda é responsável pelo armazenamento de informação sobre os dispositivos que o utilizador cria na aplicação para ligar ao Raspberry Pi. A classe App tem uma associação de um para infinito com estas duas classes pois cada utilizador pode criar um número de dispositivos indefinido bem como tirar as fotografias que desejar.

A partir da classe Devices surgem duas novas que são complementares e aumentam o número de funcionalidades do sistema no que diz respeito às tarefas com os dispositivos ligados ao Raspberry Pi. Estas duas classes são a Rules e a Events como representa o diagrama.

A classe Rules é uma generalização de duas classes, que são as implementadas no momento da programação da aplicação:

- IFRules. Esta classe pretende representar no *software* as regras condicionais criadas pelo utilizador. De notar que estas regras apenas podem ser criadas entre dispositivos, ou seja, pode ser definida qualquer regra de ativação ou desativação de um dispositivo, partindo do estado atual de ele mesmo ou outro qualquer dispositivo.

- ScheduledRules. Esta classe, complementar à IFRules, tem como objetivo fornecer dados e funções associadas com a criação e manipulação das regras temporais parametrizadas pelo utilizador. Estas regras desempenham a função de ativar ou desativar a uma determinada hora e data, sendo que esta pode ser repetida diariamente, semanalmente ou mensalmente, conforme desejo do utilizador.

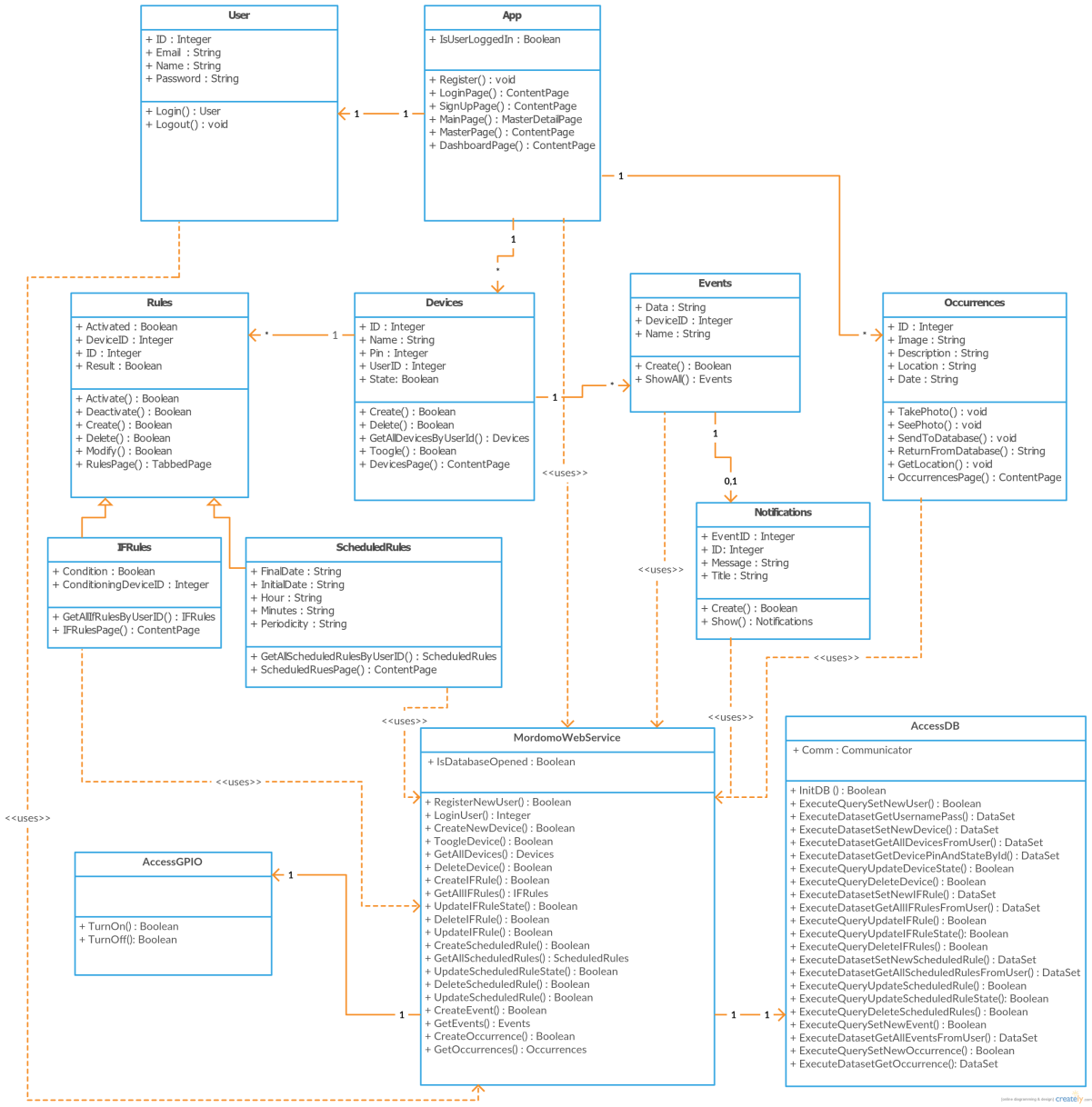

Figura 3.7: Diagrama de classes da aplicação móvel

É notório que para definição de regras o utilizador deve previamente associar à sua conta pelo menos um dispositivo.

Por fim, temos a classe Events, associada à classe Notifications. Estas duas classes representam a área do *software* destinada ao aviso do utilizador para eventuais trocas de estado dos

aparelhos, execução de regras, entre outros. Um objeto pertencente à classe Events pode suscitar, ou não, um objeto da classe Notifications, mediante o tipo de aviso que é para fornecer ao utilizador.

Todas as classes anteriormente descritas necessitam de recorrer à base de dados e/ou à interface GPIO do Raspberry Pi. Assim sendo, as classes têm todas uma associação com a classe MordomoWebService, esta que é implementada no Raspberry Pi, ao contrário das anteriores, e disponibilizada via *web service*. A classe MordomoWebService faculta funções de acesso à base de dados para a aplicação mas não acede à base de dados diretamente e, por isso, requer uma ligação com a classe AccessDB que executa os métodos para obter informação da base de dados, tal como acontece com a classe AccessGPIO para interagir com a interface GPIO do Raspberry Pi.

3.5.1.2 Interface

A interface da aplicação com o utilizador é a principal responsável para a apresentação de informação ao mesmo, e para permitir ao utilizador executar tarefas na aplicação móvel.

Na realização deste projeto optou-se por assumir uma interface simples, com controlos simples e intuitivos para os utilizadores, desvalorizando o aspeto gráfico, ou seja, nomeadamente no seu jogo de cores, entre outras características estéticas; nesta componente gráfica, o único cuidado tido em conta foi manter a aplicação coerente ao longo de todas as suas páginas, no que diz respeito as tamanhos de letra, cores e controlos.

A figura 3.8 ilustra o esquema da interface da aplicação: as diferentes páginas existentes e as ligações presentes entre si.

Num primeiro momento, o utilizador é confrontado com uma página de autenticação. Nesta página de autenticação o utilizador ou se autentica e acede às funcionalidades da aplicação, ou acede a uma outra página onde é possível se registar.

Após a autenticação ou registo, o utilizador é reencaminhado para a página chamada DashboardPage; nesta página o utilizador tem acesso a todos os eventos que ocorreram com a indicação da data e do dispositivo correspondente. Esta página, como é possível ver através do esquema referido anteriormente, tem ligação com todas as outras páginas com fins funcionais na aplicação.

É ainda relevante referir que nas páginas DevicesPage e RulesPage, dividida em duas páginas cada uma referente a um tipo de regras implementadas, o utilizador tem possibilidade de criar novos elementos, ou seja, no DevicesPage ao utilizador é-lhe facultada a possibilidade de inserir novos dispositivos, e na RulesPage o utilizador é capaz de criar novas regras para os dispositivos atualmente ativos.

O controlo para saída da aplicação está presente no menu de navegação entre as páginas. No próximo capítulo serão apresentadas figuras das páginas criadas, e todas terão associada uma descrição detalhada com as funções e toda a lógica dos procedimentos, ficando mais clara esta navegação.

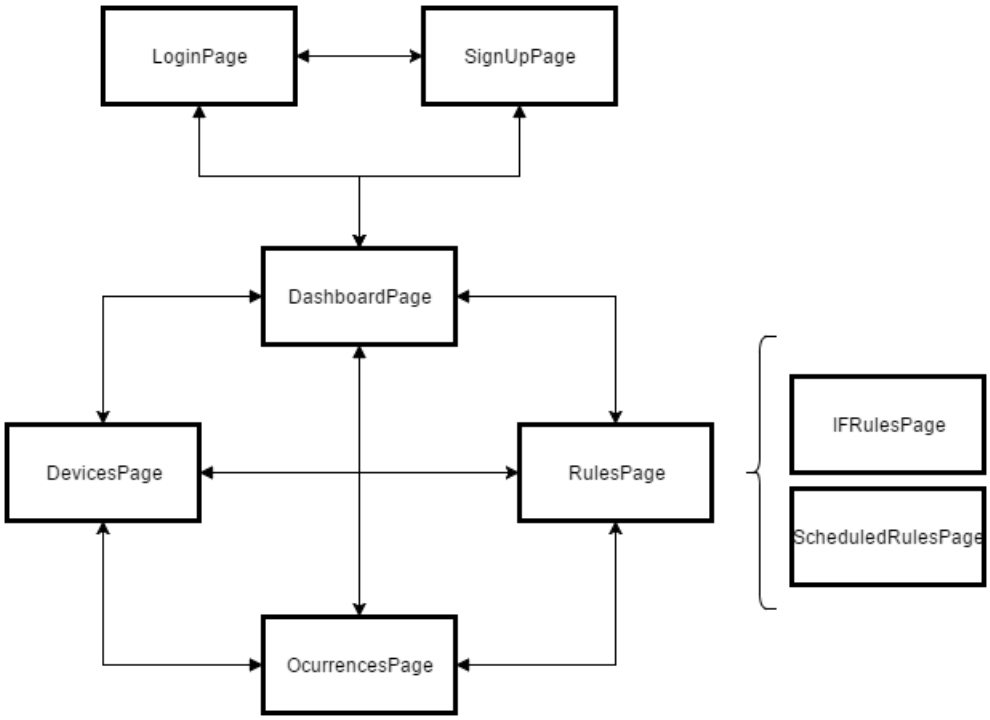

Figura 3.8: Esquema da interface da aplicação

3.5.2 Controlo autónomo

O Raspberry Pi aparece neste projeto como complemento à aplicação móvel. Apesar de o principal objetivo se prender com o desenvolvimento da aplicação, é fundamental a existência de um computador com as especificações do Raspberry Pi para assim poder finalizar uma prova de conceito. Neste sentido, e como já foi referido previamente, o Raspberry Pi é responsável pela execução de duas grandes partes do sistema: primeiro a aplicação de controlo autónomo, permitido através da verificação de regras e da comunicação com a base de dados e com a interface GPIO, realizada através do *web service*, e em segundo o próprio *web service*. Este capítulo visa estudar essencialmente a arquitetura da aplicação de controlo autónomo e por isso as restantes partes da aplicação, isto é, a camada de acesso à interface GPIO e à base de dados, não irão ser relevantes.

Com o intuito de facilitar a compreensão do funcionamento simultâneo destas duas partes, a figura 3.9 ilustra as diferentes fases do controlo autónomo e como este usufrui do *web service*.

A aplicação de controlo autónomo funciona em *loop*, ou seja, executa todos os seus passos pela ordem indicada sem cessar.

No recolher da informação, a aplicação comunica com o *web service* executando métodos disponibilizados para obtenção de dados da base de dados; é nesta fase que a aplicação lê sobre as regras parametrizadas pelo utilizador e o estado dos dispositivos, e guarda a informação nos seus atributos internos; ainda nesta fase, a aplicação compara as regras existentes atualmente na base de dados e as guardadas internamente para verificação de adição ou remoção de regras, e atualiza

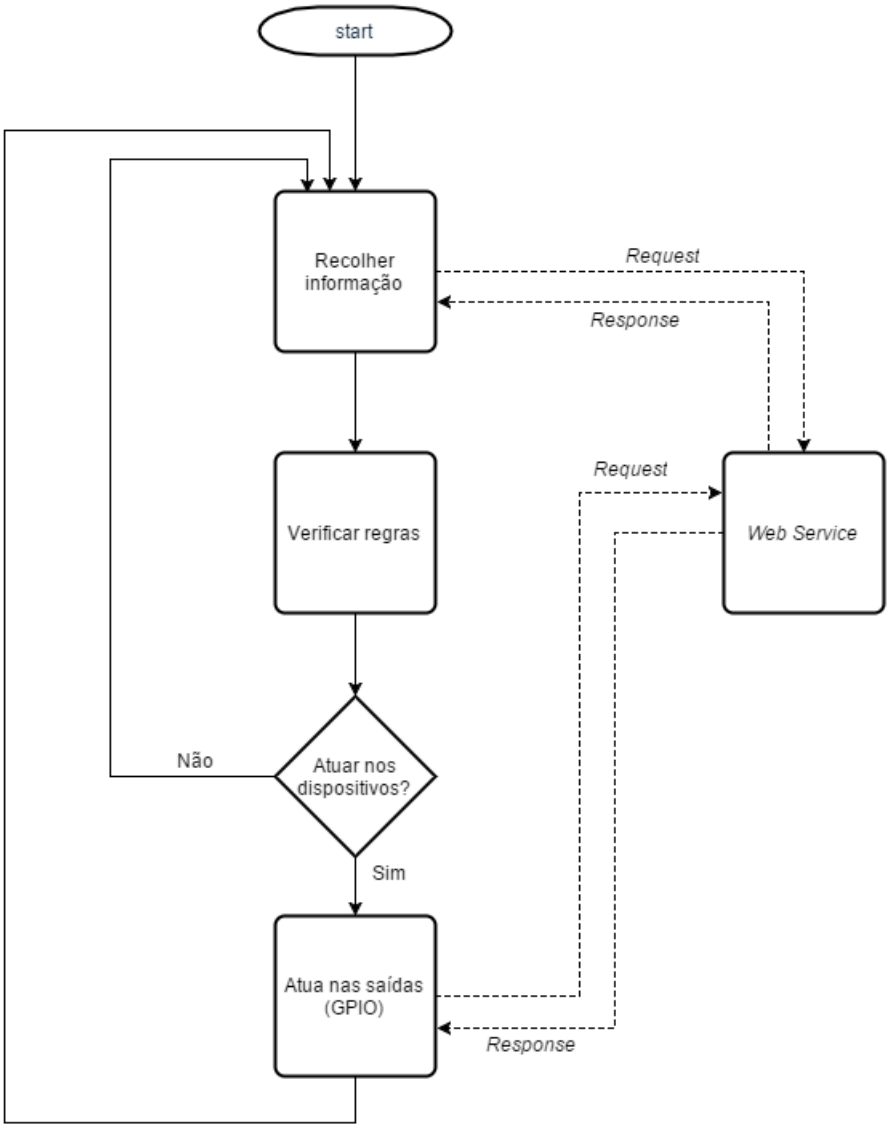

Figura 3.9: Fluxograma da aplicação do Raspberry Pi

os dados internos.

Na verificação de regras, a aplicação preocupa-se em comparar os valores atuais dos dispositivos ligados com as regras ativadas. Conforme o estado e as regras aplicadas, estas vão despoletar a atuação nas saídas na GPIO. No momento de atuar nas saídas, a aplicação volta a comunicar com o *web service*. Desta vez, os métodos evocados são os disponibilizados para ativação ou desativação de dispositivos.

3.5.3 Base de dados

A base de dados da arquitetura da aplicação é o ponto central de armazenamento de informação. A utilização de uma base de dados garante que todos os dados do sistema estão guardados de forma segura, recorrendo à obrigação de autenticação no momento inicial da comunicação. A

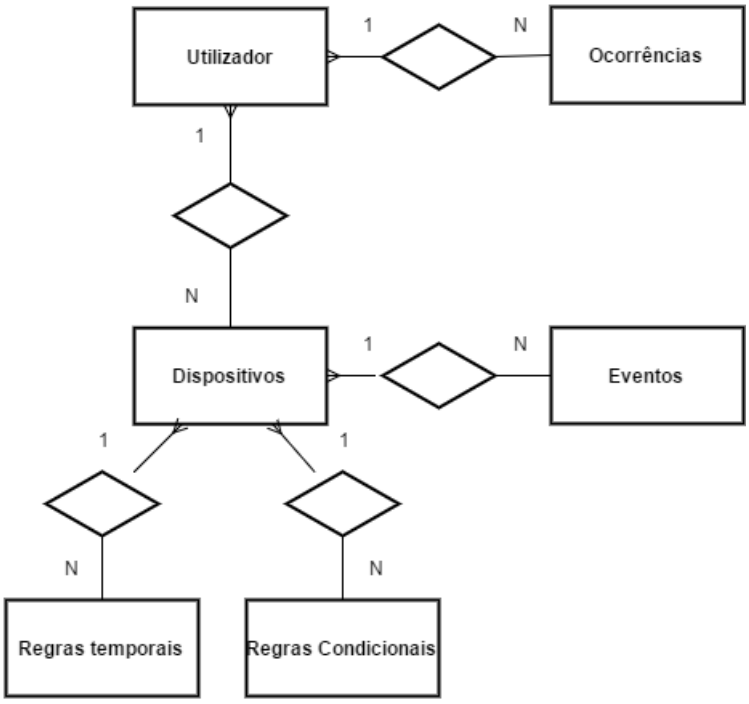

Figura 3.10: Modelo entidade-associação da base de dados

base de dados apresenta-se assim como elemento indispensável para a construção de uma aplicação com o grau de complexidade resultante das funcionalidades exigidas neste projeto.

A base de dados utilizada foi a PostgreSQL, por ser utilizada noutros trabalhos da empresa na qual o projeto foi desenvolvido, bem como por ser facilmente instalada no Raspberry Pi, e de forma gratuita.

A figura 3.10 representa o modelo entidade-associação da base de dados.

De seguida, vai ser apresentado modelo relacional da base de dados pensado para este projeto, para explicar de forma mais descritiva a mesma.

Utilizadores

ID	Nome do utilizador	Palavra-passe	Email UK

Dispositivos

ID	Nome	Pino	Estado	#Userid -> Utilizadores

Regras condicionais

ID	#IDDispositivo -> Dispositivos	#IDDispositivoAlvo -> Dispositivos	Condicao	Resultado	Ativa

Regras temporais

ID	#IDDispositivoAlvo -> Dispositivos	Resultado	Datainicial	Datafinal	Hora	Minutos
Periodicidade	Ativa					

Eventos

ID	#IDDispositivo -> Dispositivos	Mensagem	Data

Ocorrências

ID	Descricao	Localizacao	Data	Foto

No modelo relacional apresentado, cada nova linha com o número correspondente de colunas identifica uma tabela criada na base de dados.

Cada uma das tabelas tem a sua chave primária identificada a negrito, sendo em todos os casos a primeira coluna escrita com o nome ID. Da mesma forma, também as chaves estrangeiras estão identificadas: estas estão representadas com um cardinal antes do nome do atributo que aponta para uma chave estrangeira, seguida da tabela para a qual esta chave aponta; em todos os casos o atributo aponta para a chave primária da tabela que está mencionada. Por último, as chaves únicas existentes em cada tabela estão identificas com a referência UK depois do nome do atributo associado, como é o caso do atributo Email na tabela Utilizadores.

Em suma, as diferentes tabelas presentes na base de dados são as seguintes:

- Utilizadores. Esta tabela é utilizada para guardar os dados relativos aos utilizadores registados no sistema Mordomo, e é fundamental durante o executar da aplicação para identificação do utilizador e consequente relação com os seus dispositivos e ocorrências.

- Dispositivos. A tabela Dispositivos é usada para guardar a informação sobre os dispositivos criados pelos utilizadores. Têm uma chave estrangeira a apontar para a tabela Utilizadores para filtragem dos dispositivos por utilizadores, e memoriza o estado atual de cada dispositivo criado bem como o pino da interface GPIO do Raspberry Pi atribuído.

- Regras Condicionais. Nesta tabela são armazenadas as informações necessárias para executar as regras condicionais no Raspberry Pi. São guardados dois dispositivos e dois estados lógicos: um par como condição e o outro par como consequência. Os dispositivos são identificados por referencia à tabela de dispositivos, permitindo também a filtragem por utilizador.

- Regras Temporais. As regras temporais são armazenadas nesta tabela, à imagem do que acontece com as regras condicionais. As diferenças residem na identificação temporal que obriga à memorização de datas, hora, minutos e periodicidade para ativação da regra.

- Eventos. Na tabela Eventos é armazenada a mensagem que surge com a criação do evento, dependendo da sua origem. É guardada a relação com o dispositivo ao qual o evento se refere, para transmissão dessa informação ao utilizador no aparecimento dos eventos na aplicação móvel.

- Ocorrências. Por fim, foi criada esta tabela, capaz de armazenar a foto tirada pelo utilizador, assim como a localização GPS captada pelo telemóvel. É relevante referir neste caso que a foto é armazenada como sequência de caracteres, num atributo da tabela do tipo *Text*, que permite guardar texto até 2.147.483.647 bytes.

3.5.4 *Web Service*

O *web service* é um *software* projetado para apoiar as comunicações entre máquinas, realizadas via rede. É descrito num formato interpretável pela máquina, nomeadamente WSDL. O transporte de dados do *web service* é feito via protocolo HTTP ou HTTPS para conexões seguras. Os dados são enviados na linguagem XML, encapsulados pelo protocolo SOAP.

A seguinte figura ilustra o encapsulamento pelo protocolo SOAP.

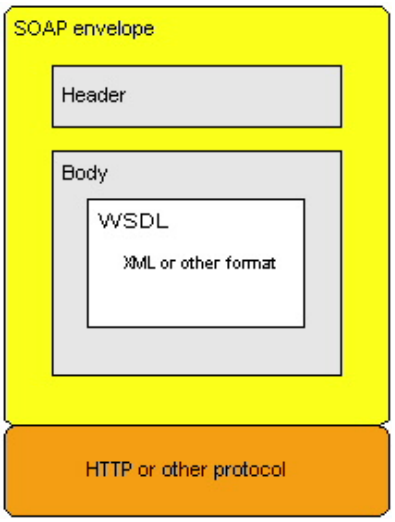

Figura 3.11: Estrutura do protocolo SOAP[13]

O consumo do *web service* exige um processo simples. O *web service* tem de estar alojado num *web server*, que por sua vez deve ser capaz de interpretar a linguagem na qual o *web service* está desenvolvido, e executar e disponibilizar o *web service*. Do lado do consumidor, é apenas obrigatório saber o endereço na rede da máquina que executa o *web service*, e todos os métodos são fornecidos com a sua respetiva descrição. A figura 3.12 representa a comunicação entre o servidor e o cliente, e excertos de mensagens em XML, segundo o protocolo SOAP.

Neste projeto especificamente, a utilização do *web service* é um ponto determinante. O sistema beneficia com o uso desta tecnologia essencialmente em dois aspetos:

1. Em primeiro lugar, com a utilização de um *web service* diversos aparelhos podem comunicar com a mesma base de dados, sem nenhum deles interferir com a informação armazenada diretamente mas apenas através dos métodos disponíveis e por isso controlados e assegurados conforme desejo do programador.

2. Por outro lado, o *web service* garante a comunicação entre a aplicação móvel e o Raspberry Pi. Ao ser utilizada uma linguagem compreendida por ambos, como é o caso do XML, é possível transmitir informação direta e facilmente entre os dois aparelhos que de outra forma seria um processo difícil ou impossível.

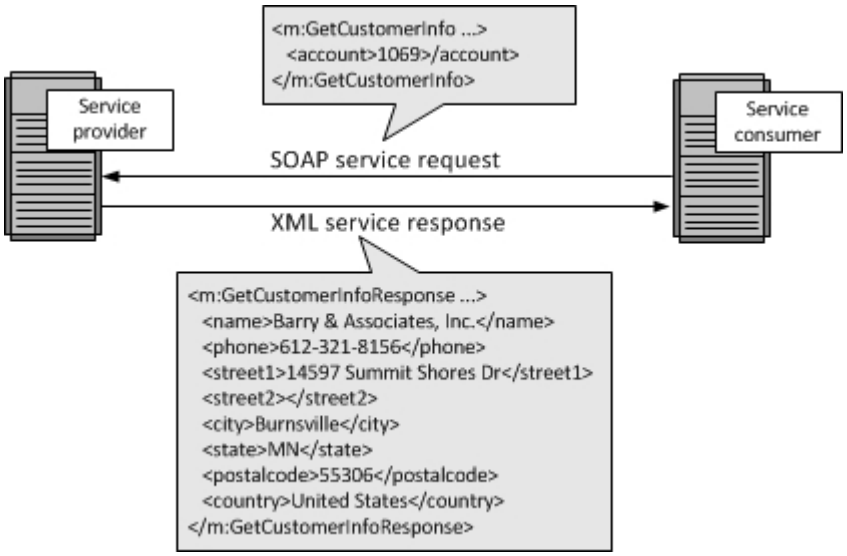

Figura 3.12: Mensagens no protocolo SOAP[13]

3.5.5 Regras temporais: diagrama de sequência

Os diagramas de sequência são, à semelhança dos casos de uso e diagramas de classes, ferramentas pertencentes à UML. Este diagrama é útil para representar a sequência de mensagens transmitidas entre os diferentes objetos existentes na aplicação. No eixo vertical está a indicação temporal, que evolui no sentido descendente; por outro lado, no eixo horizontal estão indicados os atores do processo em avaliação. De realçar que o espaço dedicado ao desenho, especialmente o vertical, não tem nenhum significado temporal relativamente ao tempo de execução da tarefa respetiva.

Este diagrama foi utilizado neste projeto para pensar e projetar o registo, armazenamento e desenrolar das regras temporais ao longo de todo o sistema, visto este ser um processo complexo e que engloba todos os integrantes do sistema Mordomo.

A figura 3.13 ilustra o diagrama de sequência da funcionalidade em estudo.

Através do diagrama, é possível verificar que o processo se inicia com a parametrização da regra por parte do utilizador através da aplicação móvel executando o método Create(). Este vai

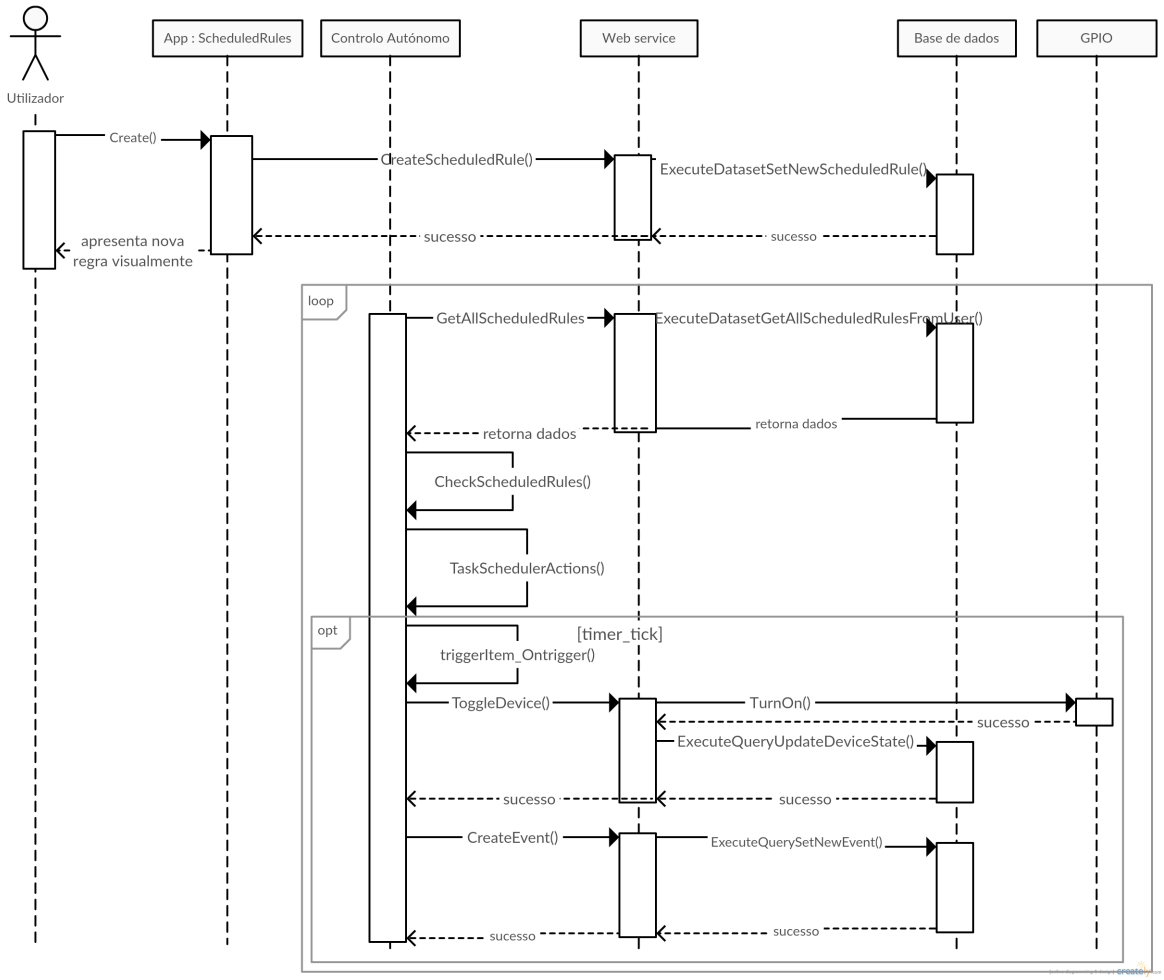

Figura 3.13: Diagrama de sequência da parametrização e execução de regras temporais

comunicar com o *web service*, que verifica as opções submetidas e comunica com a base de dados a fim de memorizar essa regra na base de dados; no retorno, é esperada uma mensagem de sucesso para informar o utilizador. Daqui para a frente não é mais necessária ação do utilizador, o sistema funciona autonomamente.

A regra é submetida aos procedimentos executados pela aplicação de controlo autónomo no Raspberry Pi, ou seja, é numa primeira fase lida através do método do *web service* e retornada para o armazenamento local do Raspberry Pi. Numa segunda fase, a regra é comparada com as já existentes e é adicionada às regras temporais atualmente ativas. Por fim, e sendo esta uma regra temporal, é incluída nos itens da classe TaskScheduler; esta classe será detalhada posteriormente no capítulo da implementação, contudo é importante referir que é a classe responsável por atribuir temporizadores às regras conforme os seus parâmetros.

Numa fase final do processo, e caso se verifique uma data e hora coincidentes com as de funcionamento da regra, as ações da regra são executadas: é ativado o dispositivo no exemplo em questão, e consequentemente alterado o seu estado na base de dados, bem como criado um evento que notifique o utilizador deste mesmo acontecimento.

Capítulo 4

Implementação

Neste capítulo são explicadas as ferramentas utilizadas na implementação do projeto para assegurar os requisitos, seguidas da demonstração dos três principais módulos que constituem o sistema cuja arquitetura foi analisada no capítulo anterior. Na apresentação dos módulos desenvolvidos, numa primeira fase será descrito o *web service*, posteriormente a aplicação de controlo autónomo presente no Raspberry Pi, e por fim a aplicação móvel.

4.1 No-IP

Um dos problemas aquando da utilização de métodos via *web service* é a variabilidade do IP da máquina responsável por alojar o serviço. No sentido de corrigir este problema, recorreu-se ao No-IP.

O No-IP é um serviço que reconhece o IP dinâmico associado à máquina na qual está instalado, e faz este IP funcionar como se fosse um IP estático, apontando um *hostname* para o IP e verificando alterações do IP com uma determinada regularidade. Assim sendo, este serviço funciona como se da atribuição do nome a um número de telefone se tratasse, ou seja, apenas é necessário conhecer o nome atribuído ao IP da máquina, independentemente das alterações que este sofra.

As principais vantagens presentes na utilização deste serviço são a facilidade com que este serviço é instalado e utilizado nos sistemas, bem como a possibilidade de aceder a uma máquina a partir de qualquer local, eliminando a necessidade de estar presente na mesma rede local.

4.2 Mono

Mono é um projeto *open-source* liderado atualmente pela Xamarin, subsidiária da Microsoft, que tem como objetivo criar um conjunto de ferramentas compatíveis com a *framework* .NET, nas quais estão inseridas o compilador C# e o CLR (Common Language Runtime).

O Mono pode ser executado a partir de vários sistemas operativos, entre os quais o Windows, Android e várias distribuições Linux, e para além de fornecer a estes sistemas operativos um compilador de C# também engloba ambientes de desenvolvimento, como é o caso do MonoDevelop.

Um dos resultados deste projeto é a plataforma Xamarin, que foi descrita anteriormente neste documento e permite o desenvolvimento de aplicações móveis para os três sistemas operativos principais no mercado.

Neste projeto em particular que está a ser descrito, para além do uso do Xamarin, o Mono é utilizado no Raspberry Pi; este programa permite executar as aplicações desenvolvidas em C# no Raspberry Pi, que funciona com o sistema operativo Raspbian (uma distribuição Linux), e desta forma aumenta a partilha de código ao longo de todo o projeto, pois as aplicações móveis são escritas em C#, de igual forma que a aplicação de controlo autónomo e o *web service*.

4.3 XSP

XSP é um *web server* escrito em C# que pode ser utilizado para executar aplicações desenvolvidas em ASP.NET. Este *web server* funciona sobre o compiladores em tempo real do Mono e da Microsoft.

O XSP é um *web server* muito básico, e é ideal para utilização em provas de conceito e pequenos testes às aplicações, não oferecendo extensibilidade nem opções de configuração. O acesso ao resultado é feito inserindo o IP ou *hostname* da máquina, seguido do número da porta na qual o *web server* lança o serviço, e podem ser observadas notificações de *debug* ou realizados testes individuais a cada uma das funções diretamente através de um browser.

Neste projeto em particular, o XSP é utilizado para executar o *web service*, desenvolvido em ASP.NET 4.0. Idealmente, para implementações em clientes é sugerida a utilização do Apache em conjunto com o mod_mono.

4.4 *Web Service*

O *web service* é a aplicação responsável por garantir a integração de todo o sistema, sendo o único meio de comunicação com o qual as outras aplicação interagem. Neste sentido, o *web service* está alojado no Raspberry Pi, e é disponibilizado recorrendo ao XSP previamente descrito.

O *web service* funciona como um servidor à espera de pedidos do cliente, isto é, são disponibilizados um determinado número de funções, denominadas *webmethods*, cada uma com um objetivo, e as aplicações externas podem aceder às mesmas através do IP da máquina onde está alojado o *web service* para utilizar os recursos do mesmo. Desta forma, às aplicações externas que recorrem às funções do *web service* apenas lhes é fornecido o resultado final, ficando a lógica interna ao *webmethod* escondida.

O *web service* comunica com duas pequenas partes da aplicação, uma delas responsável pelo acesso a dados e a outra responsável pelo acesso à interface GPIO, e para criar estas duas divisões

foram utilizadas bibliotecas auxiliares disponibilizadas em código aberto que vão ser descritas imediatamente.

Para implementar o acesso à base de dados utilizou-se a bilioteca Npgsql. A biblioteca Npgsql é um componente que dá acesso e leitura a dados armazenados numa base de dados PostgreSQL, utilizável em programas desenvolvidos em C#, Visual Basic e F#, e é totalmente escrito em C# e disponibilizado em código aberto; uma das grandes vantagens desta biblioteca é a sua compatibilidade com o Mono e o constante suporte fornecido pela comnunidade neste sentido.

Para controlar os pinos da interface GPIO utilizou-se a biblioteca raspberry-sharp-io. A raspberry-sharp-io é uma biblioteca desenvolvida pela comunidade do Raspberry Pi para a plataforma .NET, também compatível com o Mono; oferece métodos para acesso à interface GPIO, para além de ter incorporado meios para comunicar com periféricos usados regularmente para expandir as funcionalidades do Raspberry Pi, como é o caso de conversores A/D, ecrãs LCD, entre outros.

As funções desenvolvidas no *web service* existem no sentido de oferecer à aplicação móvel e a à aplicação de controlo autónomo a possibilidade de comunicar com a base de dados e com a inteface GPIO. Neste sentido, as funções apresentam internamente uma lógica de confirmação de dados submetidos pelo utilizador. As funções implementadas irão ser descritas de seguida:

- RegisterNewUser(). Este método é chamado na criação de um novo utilizador. São verificados os dados inseridos pelo utilizador para registo e inseridos na base de dados, e retorna *true* se for bem-sucedido ou *false* se ocorrer encontrar algum erro.

- LoginUser(). Este método existe para verificação dos dados introduzidos pelo utilizador. São confirmados os textos inseridos pelo utilizador é feito uma evocação à base de dados para confirmar se o par nome de utilizador e palavra-passe existem nos dados armazenados. Este método retorna o ID do utilizador autenticado.

- CreateNewDevice(). Esta função é responsável pela criação de um novo dispositivo. São filtrados os dados inseridos pelo utilizador, que é apenas o nome do dispositivo a adicionar, e é inserido este valor na base de dados para futura utilização, bem como associado o respetivo pino. O dispositivo é por defeito inicializado com valor *false*.

- ToggleDevice(): Este método tem duas partes distintas: num primeiro momento analisa qual o estado atual do dispositivo a alterar que recebe como parâmetro, partindo da informação da base de dados; num segundo momento, atua no pino do dispositivo utilizando as funções disponibilizadas pelo acesso à interface GPIO.

- DeleteDevice(). Este método recebe como parâmetro o nome do dispositivo a eliminar e atua na base de dados de forma a apagar a informação relativa ao dispositivo, suas regras e eventos associados; no caso de estar ativo, o pino do dispositivo é desligado.

- GetAllDevices(). Este método é utilizado para retornar todos os dispositivos associados a um determinado utilizador, daí retornar um vetor do tipo de classe Devices.

- CreateIFRule(). Este método é utilizado para criar uma nova regra condicional. Recebe como parâmetros os nomes dos dispositivos, determina o seu ID e memoriza na base de dados os ID's a par das condições lógicas: *true* ou *false*. Retorna valor booleano indicando sucesso ou insucesso.

- GetAllIFRules(). Este método retorna todas as regras condicionais criadas por um determinado utilizador armazenadas na base de dados.

- UpdateIFRuleState(). Esta função serve para ativar ou desativar a regra em questão. Recebe como parâmetro o ID da regra em questão, e altera o estado da regra para ativada ou desativada na base de dados.

- CreateScheduledRule(). É similar à função CreateIFRule(), mas executa o procedimento para as regras temporais.

- GetAllScheduledRules(). É idêntica à função GetAllIFRules(), mas executa o procedimento para as regras temporais.

- UpdateScheduledRuleState(). É similar à função UpdateIFRuleState(), mas executa o procedimento para as regras temporais.

- CreateEvent(). É um método implementado para criar um evento na base de dados. Recebe como parâmetros o texto referente ao evento, bem como o ID do dispositivo relacionado com o evento.

- GetEvents(). Recebe como parâmetro o ID do utilizador e envia os dados referentes aos eventos desse utilizador. Para isso com a ajuda da base de dados, associa o ID dos dispositivos dos eventos aos seus nomes para enviar.

- CreateOccurrence(). Recebe como parâmetros a foto, a localização GPS e o ID do utilizador, insere a data e fornece essa informação toda à base de dados para armazenamento. Retorna *true* ou *false* conforme sucesso ou insucesso na execução do método.

- GetOccurrences(). Retorna a última ocorrência submetida para verificação de dados na aplicação móvel. Idealmente a gestão das ocorrências seria efetuada numa aplicação destinada ao *back-office* do sistema, que não foi considerado neste projeto.

4.5 Aplicação de controlo autónomo

A aplicação de controlo autónomo é responsável pela constante verificação das regras implementadas pelo utilizador, a fim de interagir com os dispositivos associados na interface GPIO, cuja lógica de funcionamento foi explicada anteriormente no capítulo 3, na arquitetura da aplicação através da figura 3.9 . Esta aplicação é executada no Raspberry Pi, e utiliza o *web service* com dois fins: numa primeira instância, recorre ao *web service* para recolher informações da base de dados, relativas aos dispositivos e às regras implementadas; num segundo momento, e caso se verifique as condições para tal, utiliza o *web service* para modificar o estado de algum dispositivo na interface GPIO.

Para conseguir implementar esta aplicação recorreu-se a uma biblioteca externa, chamada Task Scheduler, essencial para completar o controlo das regras temporais. A biblioteca Task Scheduler é uma classe que permite agendar e automaticamente ativar um determinado evento no tempo especificado. Fornece um grande número de opções como eventos únicos, diários, semanais, mensais, e permite conjugar estas opções com o dia da semana e a semana do mês desejados. Tem funções adicionais para determinar a data da próxima ativação do evento agendado, e permite ainda memorizar os eventos agendados no formato XML, bem como ler a partir do formato XML. Esta biblioteca está disponível segundo licença aberta do repositório de partilha Code Project.

A aplicação é constituída por diversas funções e estas irão ser descritas pela ordem com a qual são executadas no Raspberry Pi. As funções são as seguintes:

- GetInfo(). Nesta função é estabelecida a comunicação com a base de dados com o objetivo de conhecer as regras atualmente memorizadas, com os seus parâmetros e condições de ativas ou não, bem como os dispositivos ativos e o seu estado atual. Essa informação é ainda guardada na memória interna da aplicação.

- CheckScheduledRules(). Este método é executado de seguida, para verificação das regras temporais. São comparadas todas as regras a fim de conhecer quais as novidades e quais as regras eliminadas; a comparação é efetuada entre as regras recém-chegadas à memória da aplicação e as regras previamente adicionadas à mesma. A porção de memória responsável por guardar as regras permanentemente é atualizada conforme as novas adições e remoções. São ainda indicadas quais as adicionadas e removidas nesta execução da rotina para futuras operações.

- CheckIFRules(). Este método apresenta uma lógica semelhante ao anterior, mas é executado para as regras condicionais.

- TaskSchedulerActions(). Neste método são adicionadas e removidas da classe Task Scheduler as regras conforme informações guardadas previamente. É feita toda a parametrização conforme o tipo de regra, ou seja diária, semanal ou mensal. São ainda criados os *triggers* para os eventos adicionados, que são em futuras operações automaticamente atualizados.

- triggerItem_OnTrigger(). Esta função é a que desempenha a lógica aquando do *trigger* da regra temporal; é ativado ou desativado o dispositivo correspondente e, no caso de existir, é criado o próximo *trigger* para o evento.

- ConditionActions(). Este método considera as regras armazenadas, e compara o estado atual com o estado da condição dos dispositivos de condição. Por fim, caso a comparação seja verdadeira, atua nos dispositivos definidos como resultado, caso o estado atual do mesmo não coincida com o desejado, a fim de evitar evocações do *web service* desnecessários.

Esta aplicação implementa ainda uma função fora da rotina executada em ciclo, que visa a deteção de botões para alteração do estado de um dispositivo. Neste sentido, é associado um botão a uma porta da interface GPIO do Raspberry Pi, que funciona com lógica *pull-up resistor*.

A lógica *pull-up resistor* está ilustrada na figura 4.1.

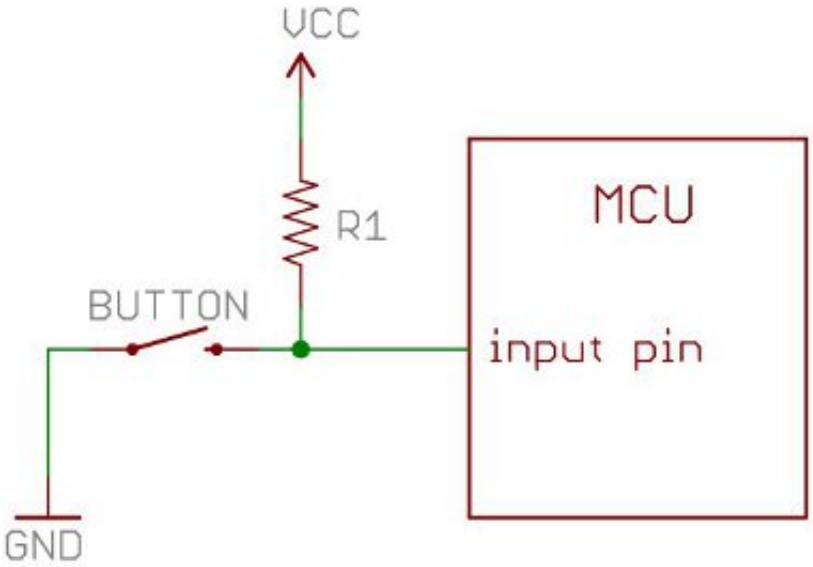

Figura 4.1: Esquema do funcionamento da lógica *pull-up resistor*[14]

Nesta lógica, o pino está sempre a detetar o valor lógico 1, associado a um nível de tensão de 3.3 volts no outro extremo da resistência R1 que provoca a existência de uma corrente entre o VCC e o pino de entrada. No momento em que o botão é pressionado, o extremo da resistência R1 que estava ligado ao pino de entrada passa a estar ligada diretamente ao *ground*, que apresenta tensão igual a 0 volts; desta forma, a corrente fluí do ponto VCC até ao *ground*, e o valor lógico do pino de entrada passa a ser 0, associado a um nível de tensão de 0 volts.

A função responsável por detetar o evento de alteração de estado do pino é disponibilizada pela biblioteca raspberry-sharp-io mencionada previamente. Nesta função, ao detetar a alteração do estado do pino para o nível lógico *false* ou 0 é chamado o *web service* para atuação no dispositivo associado a este botão; o botão funciona para alteração do estado atual do dispositivo.

4.6 Aplicação móvel

A aplicação móvel é a parte mais importante deste projeto. A interação entre o sistema e o utilizador é feita através da aplicação móvel, e assim sendo esta deve disponibilizar métodos para que o utilizador possa executar os procedimentos desejados, para além de garantir a integridade da informação submetida pelo utilizador.

Para finalizar os seus métodos, é sempre necessário o armazenamento na base de dados ou a interação com os dispositivos adicionados. É com este objetivo que a aplicação móvel comunica com o *web service*, que disponibiliza as funções para aceder à base de dados e à interface GPIO.

A aplicação móvel foi desenvolvida utilizando a plataforma Xamarin, e foi criado um projeto do tipo *shared projects*, visto isto ser uma aplicação de pequena dimensão, com poucas diferenças entre sistemas operativos realizadas com recurso a *if's*.

A interface projetada foi implementada tanto em Android como em iOS. Neste sentido, a plataforma Xamarin está munida com uma API chamada Xamarin.Forms, que disponibiliza os elementos necessários básicos para criar páginas compatíveis entre sistemas operativos, e que foi utilizada na realização deste projeto. Os elementos estão divididos em três grupos:

1. Pages. Este grupo de elementos representa os diferentes tipos de páginas que podem ser criados, que definem a apresentação a um nível elevado, e permitem maioritariamente implementar a navegação entre as páginas.

2. Layout. Os constituintes deste grupo são itens que atuam dentro das páginas definidas através dos elementos do grupo Pages; são meios para estabelecer a posição dos controlos que irão aparecer na página.

3. Controls. Por último, este grupo disponibiliza todos os controlos utilizados para interação com o utilizador; são integrados dentro dos elementos do grupo Layout que organizam a página.

A imagem 4.2 retrata os elementos gráficos da API Xamarin.Forms.

De seguida serão apresentadas as várias interfaces desenvolvidas para a aplicação e a forma de navegação e de acesso a cada funcionalidade nas mesmas, com uma descrição do método de desenho da página, para além das funções disponibilizadas em cada interface para o utilizador.

Figura 4.2: API Xamarin.Forms[15]

4.6.1 Autenticação

As páginas de autenticação apresentam essencialmente espaços para entrada de dados, quer para efetuar o login, quer para realizar o registo no sistema.

A figura 4.3 mostra dois *screenshots* das páginas construídas. Ambas as páginas são do tipo *ContentPage*.

A primeira página destina-se ao login, e é construída ao mais alto nível com cinco *StackLayout* dentro de um *StackLayout*, denominado *mainLayout*. O primeiro apresenta apenas uma figura representativa da aplicação, o segundo e terceiro são constituídos por uma imagem que indica o tipo de informação que se deve inserir nas caixas de texto imediamente a seguir, e os últimos dois *StackLayout* oferecem cada um deles um botão: o primeiro para efetuar login, submetendo os dados inseridos nas caixas de texto do tipo *Entry* para confirmação, e o último para navegação para a página de registo; de salientar que a página de registo pode ser alcançada de duas formas, sendo a primeira através do botão já referido, e a segunda através do arrastamento da página do login para a esquerda.

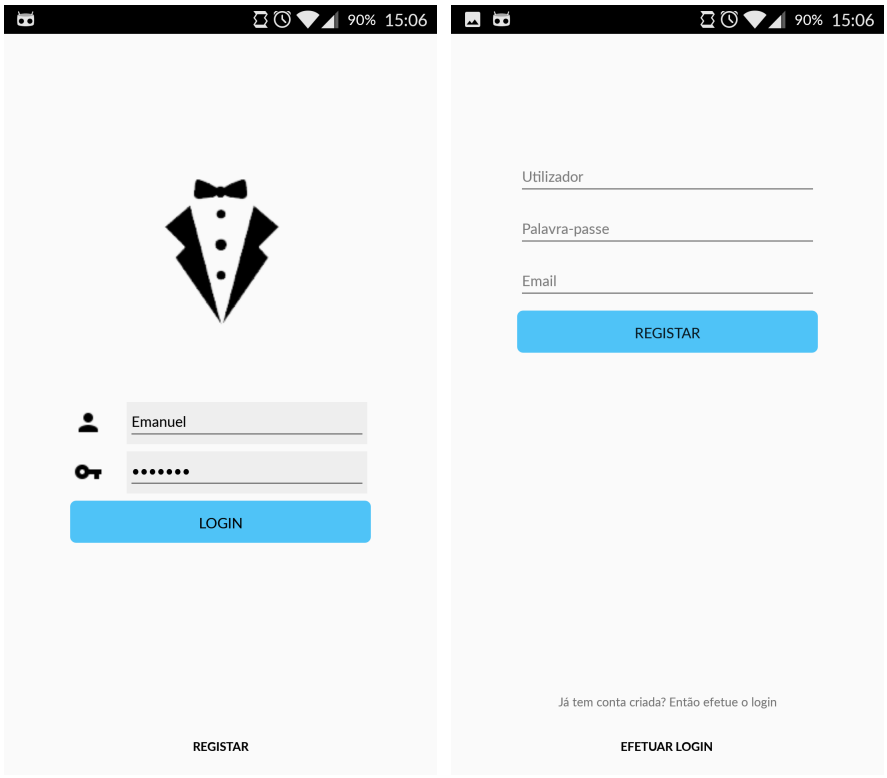

Figura 4.3: Páginas relativas aos métodos de autenticação

Na página de registo, temos acesso a três controlos do tipo *Entry*, os quais servem para inserir os dados desejados para utilização do sistema. Por fim, existem dois botões, o primeiro com o intuito de submeter os dados para verificação e criar a conta de utilizador, e o último para retroceder à página de login da aplicação móvel.

De referir que os dois botões para submissão de dados, ou seja o botão "Login"e botão "Registar", invocam os métodos respetivos do *web service*, que foram descritos já neste capítulo, enviando como parâmetros a informação inserida.

4.6.2 Dashboard

Neste subcapítulo serão apresentadas duas páginas: a página denominada Dashboard, responsável por apresentar os eventos ocorridos no sistema ao utilizador, e o menu de navegação da aplicação móvel. As páginas estão ilustradas na figura 4.4.

Do lado esquerdo, é possível visualiar a página Dashboard. Esta página é a que sucede quando o login ou o registo são efetuados com sucesso. Visualmente, a página apresenta uma sucessão de eventos que foram provocados; estes eventos são apresentados recorrendo a uma disposição gráfica criada para alocação das diferentes informações, que são a mensagem do evento, a data e o dispositivo associado, que é repetida na renderização da página. Assim, no momento de criação da página, existe um *StackLayout* que é preenchido dinamicamente, pela disposição gráfica criada,

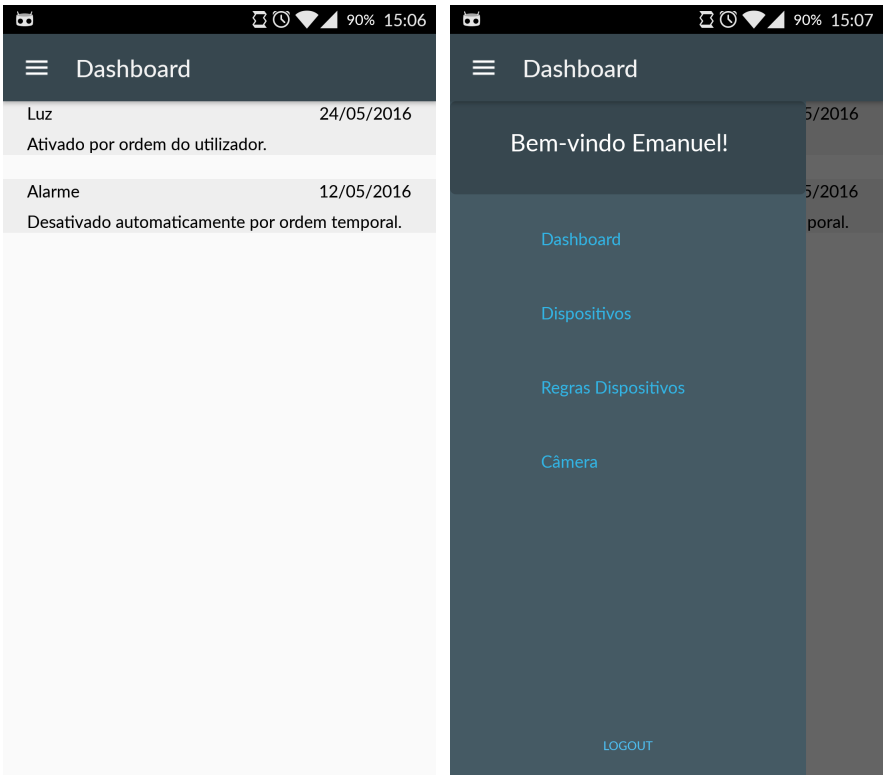

Figura 4.4: Página Dashboard e menu de navegação da aplicação

conforme o número de eventos registados até ao momento. Esta página recorre ao *web service* para obter todos os eventos memorizados na base de dados.

No lado direito da figura, está exemplificado o menu de navegação criado. Este menu é criado recorrendo a uma página do tipo *MasterDetailPage*, que é constituída pela junção de duas páginas a todo o momento:

- Uma das páginas é visível em todo o ecrã, que na figura 4.4 é a página Dashboard, que apresenta as funcionalidades. Essa página da *MasterDetailPage* é dinâmica, ou seja, depende da opção escolhida no menu de navegação.

- O segundo elemento é o menu de navegação é igualmente uma página, mas que é apenas visível em cerca de 80% do ecrã e é estática, ou seja, é gerada num momento inicial após o login e mantém-se sempre igual.

Na *MasterDetailPage* é ainda colocado um botão na parte final da página do menu de navegação que permite ao utilizador efetuar o logout da aplicação móvel.

4.6.3 Dispositivos

Na página Dispositivos são apresentados todos os dispositivos associados à conta do utilizador e disponibilizadas as funcionalidades relativos aos mesmos.

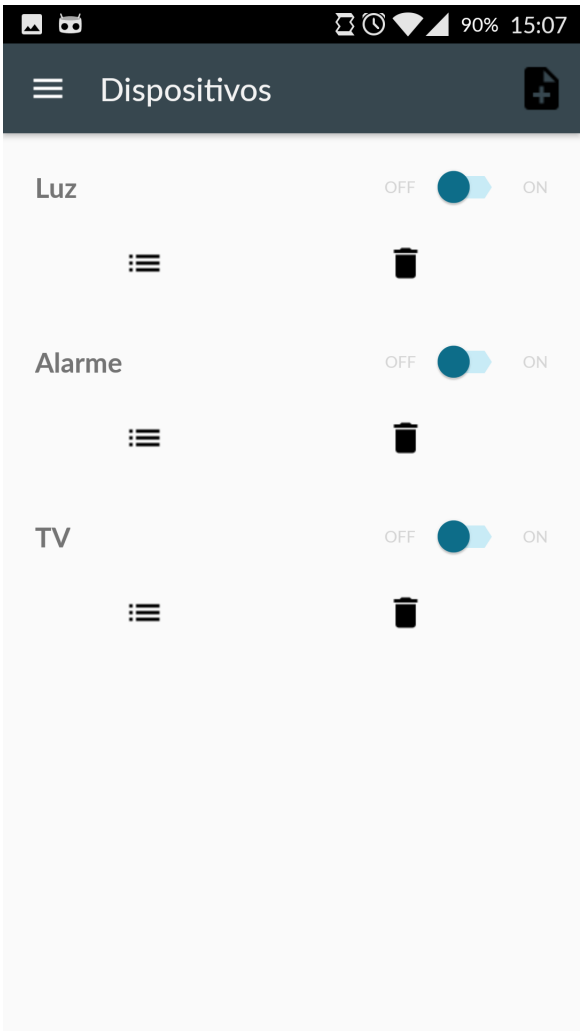

Figura 4.5: Página dispositivos

Na construção da página, num primeiro momento é, tal como acontece na página Dashboard, prepada uma aparência que depois se repete para cada um dos dispositivos. Esta aparência contém o nome do dispositivo, um controlo *Switch*, e depois botões: um para visualizar os detalhes do dispositivo e outro para eliminar o dispositivo. O botão *Switch* é aquele que permite alterar o estado do dispositivo; assim sendo, no momento da alteração do estado, provocada pelo clique do utilizador, a aplicação móvel deteta essa alteração e invoca o *web service* no sentido de alterar o estado do dispositivo. É de referir que a posição do botão *Switch* é sempre coincidente com o estado atual do dispositivo: *true* ou *false*. À imagem do que acontece com o botão *Switch*, o botão eliminar também utiliza o *web service*, a fim de eliminar o dispositivo respetivo, enviando como parâmetro o nome do dispositivo.

Para além dos elementos criados dinamicamente conforme o número de dispositivos para preenchimento da página, também esta contém no canto superior direito um botão representado por uma imagem que permite a adição de um novo dispositivo à conta do utilizador. Assim como para o botão de ver os detalhes do dispositivo, o efeito de clicar no botão de adição está representado

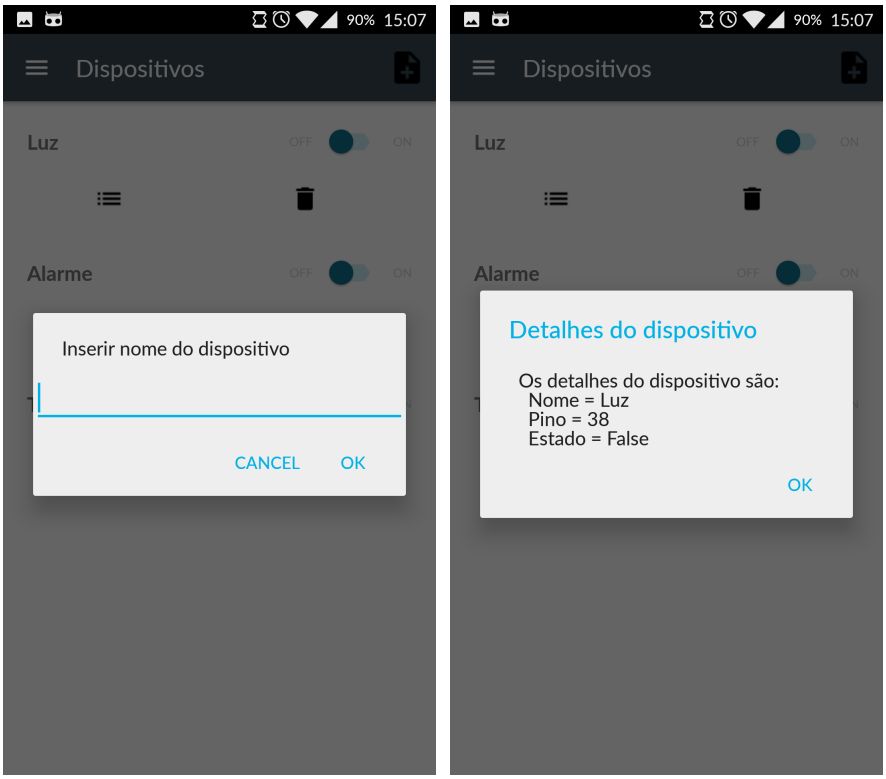

Figura 4.6: Variantes acessíveis partindo da página Dispositivos

na figura 4.6.

Primeiramente, temos o aspeto da página Dispositivos quando clicado o botão de adição de um novo dispositivo. É gerada uma caixa de alerta, para ser preenchida pelo utilizador com o nome desejado para o dispositivo. Posteriormente, é invocado o *web service* para confirmação dos dados submetidos e inserção do dispositivo na base de dados.

Seguidamente, temos a variante da página para visualização de detalhes de um determinado dispositivo. É apresentada uma pequena caixa de alerta com o nome do dispositivo e informação relativamente ao estado atual na interface GPIO, ou seja, o pino ao qual o dispositivo está associado e o estado atual do dispositivo.

4.6.4 Regras Condicionais

A página das regras condicionais é uma das páginas constituintes da página Regras de Controlo.

A página Regras de Controlo é uma *TabbedPage* constituída por duas páginas: Regras Condicionais e Regras Temporais.

Analisando a figura 4.7, a página Regras Condicionais assume dois formatos. No primeiro formato, e à imagem do processo explicado anteriormente, é gerada uma aparência que irá ser repetida ao longo do *StackLayout*, que é inserido numa *ScrollView* e que permite o arrastamento

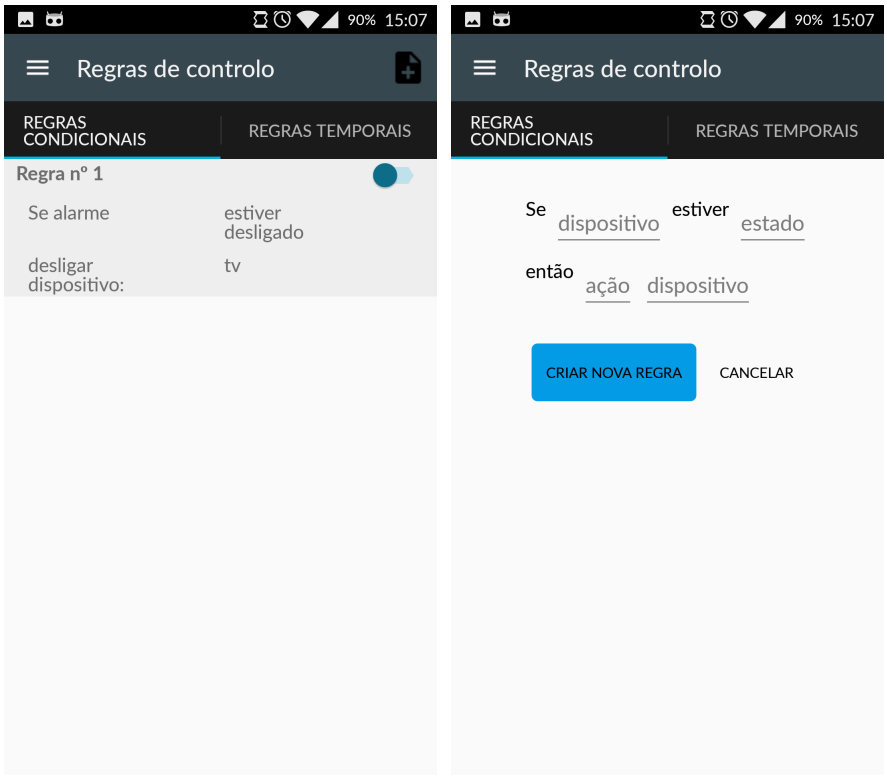

Figura 4.7: Página das regras condicionais

da página verticalmente. A aparência repetida integra a informação relativa à regra condicional criada, bem como um botão *Switch* que permite atualizar a regra enquanto ativada ou desativada.

No canto superior direito da página está presente um botão semelhante ao da página Dispositivos, que permite a criação de uma nova regra. Este botão faz alterar a informação apresentada na página, colocando assim um conjunto de elementos para o utilizador escolher os parâmetros da regra. Cada um dos elementos é um *Picker*, sendo que dois deles apresentam os dispositivos associados, outro permite escolher a ação a realizar no dispositivo a alterar, e o último escolher o estado a verificar do dispositivo que opera como condição. Um *Picker* é um controlo que abre uma caixa de alerta, oferendo movimentação vertical conforme o número de opções disponíveis. Quando os elementos são escolhdos, é pressionado o botão "Criar Nova Regra"que utiliza o *web service* para confirmar os dados e armazenar a regra na base de dados.

4.6.5 Regras Temporais

A página das regras temporais funciona de forma similar à página de regras condicionais, como é visível na figura 4.8.

A página apresenta uma primeira interface onde tem as regras temporais representadas através de uma aparência repetida para cada uma delas, a par do botão para adição de uma nova regra.

Na interface para adição de uma nova regra, tem conjunto de *Pickers*, para escolher o estado a

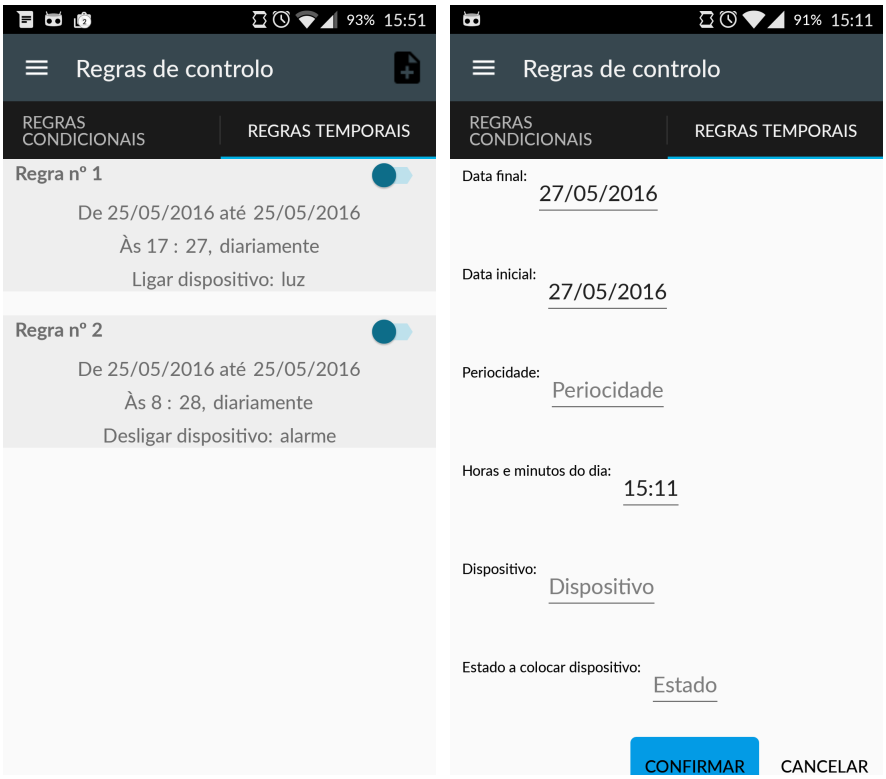

Figura 4.8: Página das regras temporais

colocar o dispositivo, a periodicidade e o dispositivo no qual atuar, tem um conjunto de *DatePickers* que possibilita a escolha de datas com seleção do dia, mês e ano, e tem por fim um *TimePicker* que permite escolher a hora do dia para atuar no dispositivo. O botão "Confirmar"invoca o *web service* para memorizar a regra temporal na base de dados e esta é também imediatamente adicionada no grupo de regras apresentadas.

4.6.6 Ocorrências

A interface apresentada na figura 4.9 tem como objetivo mostrar como é possível executar as funcionalidades associadas à operação de tirar uma foto com localização GPS, requisitos estes adicionados durante a execução do trabalho.

Para explicar o funcionamento da página, é fundamental falar de um projeto chamado Xlabs.Forms. O Xlabs.Forms é um projeto disponibilizado em código aberto que fornece diversos elementos de elevada relevância no desenvolvimento de aplicações recorrendo à API Xamarin.Forms. Este projeto faculta um conjunto de controlos e serviços compatíveis com os três sistemas operativos móveis. No projeto em questão foi essencial, pois permitiu aceder à câmera e à antena GPS, elementos nativos do *smartphone*.

Na figura 4.9 podemos verificar que a interface apresenta na parte superior da página três botões. Estes botões fornecem as seguintes funcionalidades:

Figura 4.9: Página das ocorrências

- O botão da direita executa um método que acede à antena GPS e retorna a localização atual do utilizador em coordenadas geográficas: latitude e longitude. Posteriormente, com recurso a uma funcionalidade nativa do Xamarin.Forms, é possível obter a localização atual de forma descritiva, como é visualizado na parte inferior da página.

- O botão do meio serve para tirar uma fotografia. Este botão abre uma aplicação para interagir com a câmara, permite tirar uma fotografia, confirmar ou cancelar após a gravação da mesma, e memorizar os dados da fotografia na memória interna do telemóvel.

- O botão da esquerda abre uma nova interface no telemóvel, mostrando a foto tirada imediatamente antes, caso o utilizador a deseje visualizar.

No centro da página existe uma caixa de texto, que oferece a possibilidade de adicionar uma descrição escrita à ocorrência por parte do utilizador, e apenas após o texto nesta caixa ter sido alterado é disponibilizado um botão no canto inferior da página, representado por uma figura de

confirmação, que por sua vez irá invocar o *web service* para confirmação dos dados e submissão das informações na base de dados.

4.6.7 Notificações

Por fim, foi desenvolvida uma forma de notificar os utilizadores quando estes executam algumas ações.

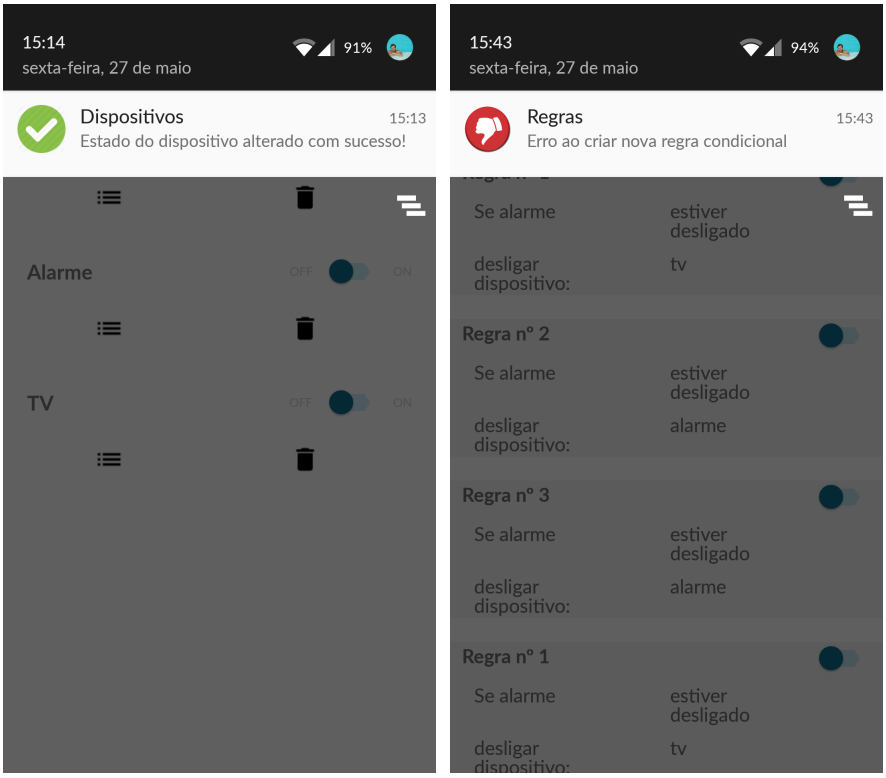

Figura 4.10: Exemplos de notificações

Em primeiro lugar na figura 4.10, é visível uma imagem que demonstra uma notificação de sucesso, sendo a imagem utilizada relacionada com tal, apresentando ainda uma pequena mensagem com data, a par da classe relacionada com a geração da notificação, que é mencionada como título.

Em segundo lugar, é mostrado um exemplo de uma notificação que informa o utilizador do insucesso na execução de alguma tarefa. As diferenças entre as mensagens de sucesso e insucesso residem precisamente na imagem associada à notificação que representa essa diferença visualmente para o utilizador.

Capítulo 5

Resultados

Neste capítulo serão apresentados os testes realizados, conforme indicados no capítulo 3, nas tabelas 3.1 e 3.2. Este capítulo pretende demonstrar a implementação descrita no capítulo 4 submetida ao plano de testes apresentado do capítulo 3.

5.1 Condições de teste

No sentido de executar o plano de testes mencionado, foi utilizado o seguinte equipamento e versões de *software*:

- Telemóvel. Foi utilizado um Oneplus One, a funcionar com o sistema operativo Android 6.0.1.
- Raspberry Pi. O modelo usado foi o Raspberry Pi 3 Model B.
- Ambiente de desenvolvimento. O projeto foi desenvolvido com recurso ao IDE Visual Studio 2013 Ultimate.
- Xamarin. Foi utilizada a API Xamarin.Forms, na versão 2.3.0.49, com a plataforma Xamarin na versão 4.0. O projeto Xlabs.Forms utilizado pertence à versão 2.0.5782.
- Mono e XPS. Alojados no Raspberry Pi, estavam o XSP na versão 4, a par do Mono na versão 4.0.5.

5.2 Testes executados

A descrição de cada teste será dividida em três fases essenciais: primeiramente a arquitetura adotada para o sistema a fim de executar o teste, em segundo lugar o procedimento realizado no teste, e por fim o resultado obtido no teste comparando-o com o resultado esperado.

Os testes realizados foram os seguintes:

1. Introduzir os dados do utilizador corretamente e incorretamente em tentativas seguidas e verificar os resultados obtidos em ambas.

    - Arquitetura. O sistema para teste incorpora três partes fundamentais: a aplicação móvel, o *web service* e a base de dados. A aplicação móvel recebe os valores introduzidos pelo utilizador e invoca um método do *web service* para filtragem dos dados e confirmação da existência do utilizador na base de dados. O Raspberry Pi e o telemóvel estão ligados à mesma rede local.
    - Procedimento. Abrir a aplicação móvel no *smartphone* e introduzir os dados nas caixas de texto correspondentes. Clicar no botão "Login"e esperar retorno de informação da aplicação móvel. De seguida tentar nova autenticação, mas introduzir um conjunto nome de utilizador e palavra-passe inválido.
    - Resultado obtido. A aplicação móvel mostrou a página Dashboard imediatamente após o clique no botão "Login"na primeira tentaiva, mostrando os eventos correspondentes com o utilizador autenticado.Na segunda tentativa a aplicação móvel informou o utilizador de que os dados introduzidos estão errados. O resultado obtido foi coincidente com o resultado esperado.

2. Aceder à área de estado dos dispositivos e verificar que todos os dispositivos estão descritos, inclusive o seu estado atual.

    - Arquitetura. Neste teste a aplicação móvel é o fundamental; contudo, é necessário o *web service* e a base de dados para obtenção da informação. A aplicação móvel encontra-se atualmente na página Dashboard. O telemóvel e o Raspberry Pi encontram-se ligados à rede.
    - Procedimento. Aceder ao menu de navegação e clicar no item Dispositivos.
    - Resultado Obtido. A aplicação móvel altera de página, apresentando os dispositivos adicionados pelo utilizador e o seu estado atual, com acesso a um botão para ver mais detalhe sobre o mesmo. O resultado obtido coincide com o resultado esperado.

3. Clicar no botão disponível para alterar o estado de um dispositivo na aplicação móvel.

    - Arquitetura. Neste teste, foram utilizadas quatro aplicações: a aplicação móvel, o *web service*, o acesso à interface GPIO e a base de dados. O telemóvel e o Raspberry Pi estão ligados à mesma rede local. Ligado entre o pino 38 e o *ground* do Raspberry Pi está um led para sinalizar visualmente o estado do dispositivo associado àquele pino. O pino está atualmente com o valor lógico *false*.
    - Procedimento. Na aplicação móvel abrir a página Dispostivos. Verificar que tem um dispositivo previamente criado e associado ao pino 38, e observar o estado atual do mesmo através da posição do botão *Switch* desse dispositivo. Clicar no botão *Switch*.

- Resultado obtido. O estado do led alterou-se, passando a estar iluminado. O botão *Switch* associado ao dispositivo na aplicação móvel também se alterou, simbolizando o estado ativo do pino. O led alterou de estado num período de tempo menos que 1 segundo; é importante realçar que este mesmo teste foi realizado com o telemóvel ligado às redes móveis e a velocidade de alteração do estado do led foi semelhante. O resultado obtido foi coincidente com o resultado esperado.

4. Aceder à área de visualização de regras e verificar todas as regras definidas e ativas no momento.

   - Arquitetura. Para este teste, é utilizada a aplicação móvel, mostrando atualmente a página Dashboard. Para além desta, também é necessário o *web service* e a base de dados para obtenção da informação. O telemóvel e o Raspberry Pi encontram-se ligados à rede.
   - Procedimento. Aceder ao menu de navegação da aplicação, e clicar no item "Regras Dispositivos".
   - Resultado obtido. A aplicação fornece duas abas, uma para as regras condicionais e outra para as regras temporais, apresentando em cada uma delas as regras criadas pelos utilizadores bem como a informação relativa à sua ativação. O resultado obtido coincide com o resultado esperado.

5. Aceder à página Dashboard na aplicação e visualizar os eventos com informações relativas como data e dispositivo associado.

   - Arquitetura. Para este teste, foi ligada a aplicação móvel, o *web service* e a base de dados. A aplicação móvel está atualmente na página de autenticação.
   - Procedimento. Realizar autenticação. Após autenticação ter sido bem sucedida, verificar que a aplicação móvel abre a página Dashboard e mostra os eventos relacionados com o utilizador. Notar que cada um dos eventos tem associado uma data, o dispositivo relacionado com o evento e a mensagem do evento.
   - Resultado obtido. A aplicação móvel apresentou os eventos do utilizador com as informações relevantes, e desta forma o resultado obtido coincidiu com o resultado esperado.

6. Aceder à área de definição de regras temporais e definir um horário e estado para um determinado dispositivo.

   - Arquitetura. Para este teste, o elemento fundamentai é apenas a aplicação móvel. A aplicação estará aberta na página Regras Temporais.
   - Procedimento. Clicar no botão para adicionar uma nova regra temporal no canto superior direito.

- Resultado obtido. A aplicação disponibiliza espaços para inserir os horários desejados, o estado a colocar o dispositivo, a periocidade da regra, as datas entre as quais a regra atua, bem como permite escolher o dispositivo a alterar. O resultado obtido coincide com o resultado esperado.

7. Aceder à área de definição de regras condicionais e atribuir uma condição para alteração do estado de um dispositivo.

   - Arquitetura. A aplicação móvel é a única parte relevante na execução deste teste, estando a mostrar a página Regras Condicionais.
   - Procedimento. Clicar no botão para adicionar uma nova regra condicional no canto superior direito.
   - Resultado obtido. A aplicação móvel abre uma nova aparência gráfica, disponibilizando controlos para adicionar dois dispositivos, um para verificar e outro para atuar, e duas condições, uma para verificação e outra para atuação. O resultado obtido é coincidente com o resultado esperado.

8. Após a definição de regras, verificar que o Raspberry Pi altera o estado do dispositivo em conformidade com as mesmas.

   - Arquitetura. Para este teste, são utilizadas três aplicações: a aplicação de controlo autónomo, o *web service* e a base de dados. A base de dados é acedida diretamente com o programa pgAdmin para uma manipulação mais fácil e rápida da informação.
   - Procedimento. Ligar um led entre o pino 38 e o *ground* do Raspberry Pi. Adicionar 120 regras na base de dados, com a indicação para alteração do estado pino 38 a cada minuto.
   - Resultado Obtido. Verificou-se que durante 20 minutos o Raspberry Pi alterou o estado do pino 38 a cada minuto, ligando e desligando continuamente o led. Desta forma, conclui-se que o sistema consegue funcionar corretamente com pelo menos 100 regras guardadas na base de dados, e o resultado obtido coincide com o resultado esperado.

9. Provocar a alteração de um aparelho e verificar que é automaticamente gerada uma notificação na aplicação móvel.

   - Arquitetura. Neste teste são utilizadas quatro aplicações: a aplicação móvel, a aplicação de controlo autónomo, o *web service* e a base de dados. A aplicação móvel apresenta atualmente a página dos dispositivos. O telemóvel e o Raspberry Pi encontram-se ligados à rede.
   - Procedimento. Clicar no botão *Switch* para alteração do estado do pino de um dispositivo.

- Resultado Obtido. A aplicação cria uma notificação automaticamente, avisando sobre o sucesso da alteração do estado do pino, o que indica que o resultado obtido é coincidente com o esperado. De realçar que o resultado deste teste está presente na figura 4.10, no *screeshot* da esquerda.

10. Provocar o sucesso e insucesso em determinadas tarefas e verificar que no dispositivo móvel as notificações aparecem com cor relativa ao sucesso e insucesso.

    - Arquitetura. Neste teste são utilizadas quatro aplicações: a aplicação móvel, a aplicação de controlo autónomo, o *web service* e a base de dados. A aplicação móvel apresenta atualmente a página dos dispositivos. O telemóvel e o Raspberry Pi encontram-se ligados à rede.
    - Procedimento. Clicar no botão *Switch* de um dispositivo. De seguida desligar o cabo ethernet do Raspberry Pi, e voltar a clicar no botão *Switch* de um dispositivo.
    - Resultado Obtido. A aplicação gera duas notificações: uma de sucesso e outra de insucesso, alterando a imagem entre notificações. O resultado obtido era o resultado esperado.

11. Clicar no botão disponível para ver a foto e com recurso a uma função que recolha a foto da base de dados verificar que a foto está visível para posterior uso.

    - Arquitetura. Neste teste, foi utilizada a aplicação móvel, bem como o *web service* e a base de dados; todos os elementos estão ligados à rede. A aplicação móvel está ligada na página das ocorrências.
    - Procedimento. Clicar no botão do centro de entre os disponíveis na parte superior da página. Tirar a foto e seguidamente adicionar uma descrição e confirmar. De seguida, clicar no botão da esquerda.
    - Resultado Obtido. A aplicação abre uma nova página, mostrando a foto acabada de tirar com a aplicação Mordomo e guardada na base de dados. O resultado obtido era o resultado esperado.

12. Clicar no botão disponível para obter a localização GPS e verificar que a localização indicada corresponde à localização atual do utilizador.

    - Arquitetura. Neste teste é apenas utilizada a aplicação móvel, disponibilizando a página das ocorrências.
    - Procedimento. Clicar no botão da direita na parte superior da página.
    - Resultado obtido. A aplicação móvel preenche uma caixa de texto com a informação relativa à localização atual. Essa localização atual coincide com a localização verdadeira do utilizador, ou seja, o resultado obtido coincide com o resultado esperado.

13. Executar a aplicação na mesma rede que o Raspberry Pi e a partir de outra distinta e verificar que a conexão é estabelecida em ambos os casos.

- Arquitetura. Para este teste, são usados a aplicação móvel, o *web service* e a base de dados. Num primeiro momento a aplicação móvel está ligada à mesma rede Wi-Fi que o Raspberry Pi.
- Procedimento. Executar a aplicação e realizar a autenticação; após o resultado da autenticação, desligar a aplicação móvel. Desligar o Wi-Fi nas definições do telemóvel, e ligar as redes móveis do telemóvel. Esperar que o telemóvel apresente sinal de rede e abrir pela segunda vez a aplicação móvel. Voltar a inserir os dados da autenticação e confirmar os mesmos.
- Resultado obtido. A autenticação foi bem sucedida em ambas as tentativas, confirmando que é possível utilizar a aplicação a partir da mesma rede local que o Raspberry Pi ou a partir de outra distinta, confirmando que o resultado obtido coincide com o resultado esperado.

14. Conectar dois smartphones em simultâneo ao Raspberry Pi e realizar pedidos diferentes partindo de aparelhos diferentes.

    - Arquitetura. Nesta situação foram utilizados dois telemóveis, correndo a aplicação móvel Mordomo, para além do *web service* e da base de dados. Todos os componentes estavam ligados à mesma rede local, e os *smartphones* a mostrar a página dos dispositivos, autenticados com o mesmo utilizador. O Raspberry Pi tinha ligado entre os seus pinos 38 e *ground* um led, inicialmente no estado *false*.
    - Procedimento. Clicar para alterar o estado do dispositivo num dos telemóveis e seguidamente fazer o mesmo no outro telemóvel, interagindo em ambos os casos com o mesmo dispositivo.
    - Resultado obtido. O led ligou num primeiro momento e desligou de seguida, mostrando assim que os telemóveis podem comunicar ambos com o mesmo dispositivo, sendo o resultado obtido igual ao esperado.

15. Provocar uma falha de rede no Raspberry Pi e verificar que os componentes de hardware apresentam resultados segundo as regras previamente guardadas.

    - Arquitetura. Para este teste é apenas utilizada a aplicação de controlo autónomo, sendo que a informação da base de dados é previamente conhecida e adquirida pelo Raspberry Pi. O Raspberry Pi tem novamente ligado à sua interface um led, entre os pinos 38 e o *ground*.
    - Procedimento. Criar previamente regras na base de dados e desligar o cabo ethernet do Raspberry Pi.
    - Resultado obtido. O Raspberry Pi ligou o led segundo a regra implementada, ou seja, se o led estivesse desativado ele teria de o ativar numa hora estabelecida na regra. O resultado obtido coincide com o esperado.

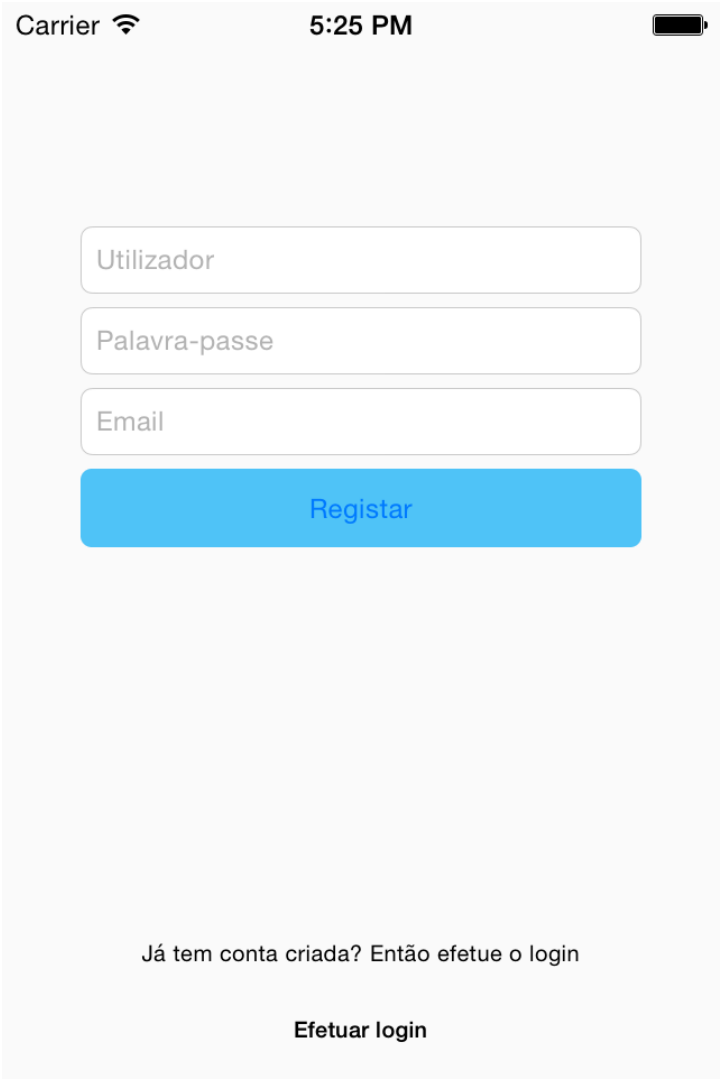

Figura 5.1: Página exemplo no emulador iOS

16. Ao executar a aplicação desenvolvida em *smartphones* com os sistemas operativos referidos a aplicação deverá funcionar disponibilizar todas as funcionalidades desenvolvidas.

    - Arquitetura. Neste teste foram necessários um *smartphone* com o sistema operativo Android e, devido à ausência de um aparelho iOS, foi utilizado um emulador num portátil Mac.
    - Procedimento. Instalar a aplicação num telemóvel Android 6.0 e executá-la. Posteriormente, correr a aplicação no emulador iOS 7.4.
    - Resultado obtido. A aplicação é executada com sucesso em ambos os casos, apesar de apresentar pequenas diferenças de aparência gráfica. O teste foi positivo mediante as condições de teste aprovisionadas. A figura 5.1 mostra um exemplo de uma página no emulador iOS.

17. Executar um comando através da aplicação, seguido de uma alteração direta através de um interruptor, e verificar a alteração do estado do componente em ambas as situações.

    - Arquitetura. Nesta aplicação foi utilizada a aplicação móvel, o *web service*, a base de dados e a aplicação de controlo autónomo. Na interface GPIO do Raspberry Pi estava ligado um led entre pino 38 e o *ground* e um botão entre o pino 12 e o *ground*, com um resistência ligada entre o pino 12 e os 3.3V.
    - Procedimento. Clicar no botão *Switch* da aplicação móvel para alteração do estado do dispositivo e seguidamente clicar no botão físico ligado ao pino 6.
    - Resultado obtido. Num primeiro momento o led ligou-se, ao clicar no botão disponível na aplicação móvel. Ao clicar no botão físico, o led desligou-se, confirmando assim que o resultado esperado foi atingido.

Capítulo 6

Conclusões

6.1 Análise e Avaliação do Projeto

O trabalho desenvolvido ao longo da dissertação contribuiu para o desenho e desenvolvimento de um sistema piloto na área da domótica. Foi essencial o estudo do mercado para compreender em que nível se encontram os sistemas comercializados pelas principais empresas nesta área, no sentido de decidir como a empresa pode iniciar e eventualmente concorrer no mercado. Outro ponto fundamental resultante deste projeto consiste na avaliação e experimentação das diferentes tecnologias utilizadas, que fornecem uma base para futuras melhorias.

A metodologia adotada mostrou-se uma escolha acertada para resolução do problema, sendo que os principais objetivos foram atingidos com sucesso. A arquitetura pensada para o sistema revelou-se um bom ponto de partida, pois cada aplicação referente a uma função dentro do sistema foi desenvolvido separadamente, o que possibilita a melhoria futura de módulos específicos, com principal atenção para o módulo de acesso ao *hardware*. As funcionalidades implementadas coincidem com a base fornecida em qualquer sistema domótico comercializado, permitindo focar o trabalho futuro na inovação e melhoria do sistema.

Em suma, o trabalho final atinge um nível bastante relevante para os interesses da empresa. Apesar de não ser viável comercializar este sistema, pois apresenta lacunas no que diz respeito à facilidade e possibilidade de implementação em espaços residenciais, as bases estão criadas para o futuro.

Do ponto de vista pessoal, este projeto mostrou-se enriquecedor permitindo a interação com tecnologias nunca antes utilizadas, como é o caso do desenvolvimento de aplicações móveis, utilização de *web services*, interação com o Raspberry Pi e sistemas operativos Linux. Possibilitou ainda um primeiro contacto com conceitos de engenharia de *software* nunca antes estudados.

6.2 Trabalho futuro

O sistema desenvolvido utiliza tecnologias que permitem uma inovação imensa para trabalhos futuros.

No que toca à camada do *hardware*, no futuro torna-se fundamental dotar o sistema com métodos capazes de interagir com um número de dispositivos maior, nomeadamente sensores e aparelhos utilizados nos espaços residenciais, como é o caso de máquinas de lavar, fogões, entre outro. A camada de *harware* é notoriamente a que necessita de maior intervenção para o sucesso do projeto.

Sobre a camada de *software* podem ser implementadas diversas melhorias, entre as quais a melhoria das regras condicionais para permitir condições múltiplas e condições baseadas na localização GPS do utilizador, assim como mecanismos para implementar videovigilância com uma ou várias câmaras. Poderia ser interessante e atrativo integrar no sistema métodos de gestão, disponibilizando para os utilizadores gráficos de consumo e/ou utilização relativos aos aparelhos associados.

Esta área é definitivamente um espaço para muitas inovações e melhorias, com enorme interesse tecnológico e que ainda não tem uma solução ideal e dominante para atração dos consumidores.

Referências

[1] K. Association. http://www.knx.org/knx-en/index.php.

[2] D. S. da Costa Palma, "Feup knx domótica knx/eib de baixo custo," Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2008.

[3] S. H. USA. http://www.smarthomeusa.com/how-x10-works/.

[4] INSTEON, *INSTEON WHITEPAPER: The Details*, 2013.

[5] Mobilefun. http://www.mobilefun.co.uk/belkin-wemo-home-automation-with-motion-bundle-for-iphone-ipad-touch-p38870.htm.

[6] SmartThings. http://docs.smartthings.com/en/latest/architecture/capability-taxonomy.

[7] myDevices Cayenne. http://www.cayenne-mydevices.com/.

[8] J. M. M. de Barros Zão, "Módulo 'o meu mordomo' para aplicações móveis e domótica," Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2015.

[9] A. Trice. http://phonegap.com/blog/2012/05/02/phonegap-explained-visually/.

[10] X. U. the Xamarin Mobile Platform. https://developer.xamarin.com/guides/cross-platform/.

[11] X. S. C. Options. https://docs.xamarin.com/guides/cross-platform/.

[12] Wikipédia. http://bit.ly/28YXaXL.

[13] S. Architecture. http://www.service-architecture.com/articles/web-services/soap.html.

[14] Sparkfun. ttps://learn.sparkfun.com/tutorials/pull-up-resistors.

[15] Xamarin. https://xamarin.com/forms.

[16] ITChannel. http://www.itchannel.pt/news/eventos/ifa-2015-a-samsung-comeca-a-dar-vida-ao-iot.

[17] F. J. da Silva Oliveira, "Rede domótica compatível com knx," Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2013.

[18] Arduino. https://www.arduino.cc/en/Tutorial/X10.

[19] eletronic design. http://electronicdesign.com/communications/what-s-difference-between-zigbee-and-z-wave.

[20] Wired. http://www.wired.com/2016/02/wemo-smart-switch-recipes/.

[21] Wired. http://www.wired.com/2015/09/smartthings-desperately-wants-turn-houses-smart-homes/.

[22] IDC. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[23] N. Savula. http://elekslabs.com/2014/04/inside-appcelerator-titanium.html.

[24] Xamarin. https://xamarin.com/customers.

[25] C. PETZOLD, *Creating Mobile Apps with Xamarin.Forms*. Microsoft Press, 2016.